L. J. Graham
T. Field

# Guida al PC-IBM



# Guida al PC-IBM

L. J. Graham
T. Field

# Guida al PC-IBM

McGRAW-HILL Libri Italia srl

**Milano** · New York · St Louis · San Francisco · Amburgo · Auckland
Bogotá · Città del Guatemala · Città del Messico · Lisbona
Londra · Madrid · Montreal · Nuova Delhi · Panama · Parigi · San Juan
San Paolo · Singapore · Sydney · Tokyo · Toronto

Da un originale Osborne/McGraw-Hill



Titolo originale: *Your IBM PC*


Realizzazione editoriale: EDIGEO srl, via del Lauro 3, 20121 Milano
Traduzione: Anna Wolter e Fulvio Toma
Grafica di copertina: Valentina Boffa
Composizione e stampa: Litovelox, Trento

ISBN 88-386-0021-X

1ª edizione ottobre 1986
1ª ristampa aprile 1987
2ª ristampa febbraio 1988

**RINGRAZIAMENTI**

Il contenuto di questo libro riflette i contributi di molte fonti: senza di esse il progetto del libro stesso sarebbe stato portato a termine con maggiori difficoltà e senza ottenere il risultato sperato.
Vorrei innanzitutto esprimere la mia gratitudine a Don Shapiro, Roger Brooks, John Sliney, Tony Hartman, Tim Holechek e Andy Coblentz, così come a Joyce M., Rob B. Anna L., Mark P., Brian D. e Clair e North Beach per il loro aiuto, ispirazione ed entusiasmo.
A Dave Wilson della *SRI International* devo molte idee ed il continuo incoraggiamento durante tutto lo svolgimento del lavoro. Il Capitolo 9, un eccellente trattato sulle possibilità grafiche del PC con il BASIC, è stato scritto da Marty McNiff e Karl Koessel. Inoltre Ralph Baumgartner, Steve Cook ed altri redattori della Osborne/McGraw-Hill hanno contribuito alla compilazione del Capitolo 12 e delle Appendici.
Il mio lavoro infine è stato reso possibile dall'eccellente e professionale staff della Osborne/McGraw-Hill.

*Lyle J. Graham*

# Indice

# Introduzione

Questo libro vi metterà in grado di capire il funzionamento del Personal Computer IBM, qualunque sia il vostro livello di conoscenza dei computer. Nel libro vengono descritti sia il PC dotato della versione 2.10 del sistema operativo PC-DOS che il PC/XT. Tutte le descrizioni, tranne quando sottolineato esplicitamente e le operazioni si adattano anche all'XT, di cui descriveremo ampiamente caratteristiche e particolari.
In questo libro sono trattati tre aspetti dell'uso del PC-IBM: dapprima le parti che possono formare un sistema, compresi i componenti periferici; in secondo luogo una lunga esposizione dei differenti modi d'uso del PC, sia con programmi applicativi che potete trovare in vendita, sia con programmi scritti da voi stessi; ultimo, e più importante, il controllo del PC attraverso i vari comandi da tastiera e suggerimenti affinché impariate a scrivere i programmi da soli.
I Capitoli 1 e 2 sono destinati a tutti gli utenti del PC. Nel Capitolo 1, "Il Personal Computer IBM", vengono descritte le parti comuni a tutti gli elaboratori della famiglia dei Personal Computer IBM e alcuni particolari dispositivi che caratterizzano specificatamente il funzionamento del PC. Viene inoltre presentato un largo spettro di applicazioni per il PC, in modo che possiate apprezzarne appieno la versatilità.
Il Capitolo 2, "Descrizione del PC-IBM", offre una visione d'insieme dell'uso del PC, senza peraltro presupporre da parte dell'utente alcuna conoscenza del computer; dopo aver terminato la lettura del Capitolo 2 sarete in grado di far eseguire al vostro personal programmi scritti in linguaggio BASIC IBM e di usare la versione 2.10 del sistema operativo PC-DOS.
Nel Capitolo 3, "Il sistema operativo", viene esaminata in dettaglio la

versione 2.10 del PC-DOS, e viene mostrato come usare questo sistema operativo ormai molto popolare.
I Capitoli dal 4 all'11 sono dedicati a quei lettori che vogliano programmare il loro PC in BASIC. Anche per la lettura di questi capitoli non è necessaria una conoscenza precedente del BASIC, ma può esservi d'aiuto l'aver avuto esperienze di programmazione.
I Capitoli 4, "Introduzione al BASIC", 5, "Fondamenti di BASIC" e 6, "Uso avanzato del BASIC", vi permettono di trovare soluzioni personali a tutti i vostri problemi, insegnandovi a scrivere programmi in BASIC. In essi viene spiegato interamente il processo di programmazione e il metodo per correggere i programmi. Troverete infine esaurienti informazioni su come sfruttare al meglio, nella scrittura dei programmi, le caratteristiche proprie del PC.
Il Capitolo 7, "I file BASIC", tratta della memorizzazione e del caricamento di dati nei vari tipi di memorie di massa attraverso diversi dispositivi di input/output. Per esempio, imparerete come si possono organizzare le informazioni per mezzo di programmi scritti in BASIC, come immagazzinarle su floppy disk o dischi rigidi e come accedervi facilmente.
La lettura del Capitolo 8, "La memoria del PC-IBM", vi sarà utile se avete già fatto qualche esperienza di programmazione in BASIC o se volete cimentarvi nell'uso di altri linguaggi con il PC; infatti vi viene mostrato lo schema di organizzazione della memoria del PC e come accedere ad ognuna delle sue parti. Inoltre questo importante capitolo illustra l'interazione tra programmi in BASIC e memoria e i metodi di controllo di questo processo da parte dell'utente.
Il Capitolo 9, "La grafica", vi insegnerà a creare con l'aiuto del BASIC figure e diagrammi sullo schermo del vostro PC; la grafica infatti ha molto spesso una parte importante nei programmi per i computer, poiché il suo uso corretto può notevolmente migliorare la comunicazione tra utente e programma.
Nel Capitolo 10, "Il suono", viene fornita una guida alla produzione, sempre in BASIC, di suoni col PC. Sarete così in grado di creare suoni, specificando frequenza e durata, oppure di trasferire musica al PC e fargliela eseguire, specificando note, tempi, lunghezza delle note e quant'altri dati siano necessari.
Il Capitolo 11, "Comunicazioni", fornisce esaurienti spiegazioni sul come mettere in comunicazione il PC con altri dispositivi, tra i quali altri PC, altri computer differenti dal PC, e in generale con diversi dispositivi di input/output. Potrete inoltre apprendere sia le tecniche di comunicazione hardware attraverso programmi BASIC, sia le informazioni di base per poter scrivere voi stessi programmi originali in BASIC che consentano le comunicazioni.
Il Capitolo 12, "Comandi DOS per lo sviluppo dei programmi", mostra l'uso di tre strumenti di sviluppo dei programmi, quando si utilizza il PC

sotto il sistema operativo DOS.
L'ultimo Capitolo, il 13, "Manutenzione del PC", dà alcuni utili consigli sul come predisporre lo spazio in cui sistemare il computer e come affrontare i problemi che possono sorgere durante l'uso del PC; viene delineata inoltre la prassi per attuare una installazione ideale del sistema e le prime procedure operative.

Capitolo

# 1 Il Personal Computer IBM

Il Personal Computer IBM, che noi siamo soliti chiamare semplicemente PC-IBM o ancora più familiarmente PC, può essere tranquillamente definito un completo sistema a computer; è composto di un numero limitato di parti fondamentali e solitamente di alcune parti aggiuntive, optional, che lo predispongono per particolari applicazioni.
In questo volume tratteremo di entrambe le versioni disponibili del PC, cioè della versione standard, nota come PC-IBM, e della versione PC/XT, dove XT sta per "eXTended", con configurazione hardware più estesa di quella della versione base. In tutto il libro, comunque, la sigla PC verrà genericamente usata per riferirsi sia alla versione standard che alla versione XT; solo qualora risultasse rilevante farlo, verrà messo l'accento sulle differenze tra le due versioni.
In questo capitolo impareremo prima di tutto di quali parti fondamentali è composto il PC e qual è il compito di ognuna di esse; in seguito porremo la nostra attenzione su alcuni dei più diffusi dispositivi opzionali che possono essere aggiunti al sistema per personalizzarlo e sulle applicazioni in cui essi trovano impiego. Si osservi infine che, quando parliamo di dispositivi, intendiamo fare riferimento sia a componenti fisici, cioè l'hardware, sia a istruzioni e programmi che hanno il compito di controllare l'hardware, cioè il software.

## 1.1 L'hardware del PC

I dispositivi hardware fondamentali che compongono il sistema PC sono:

— L'unità di sistema con la memoria
— La tastiera
— Il monitor
— Un dispositivo di memoria di massa

### L'UNITÀ DI SISTEMA

Il cuore del PC è l'unità di sistema, mostrata in Figura 1.1, contenente un circuito stampato, che prende il nome di *System Board*, o scheda di sistema, sulla quale trovano sede tutti i circuiti elettronici di base del PC.

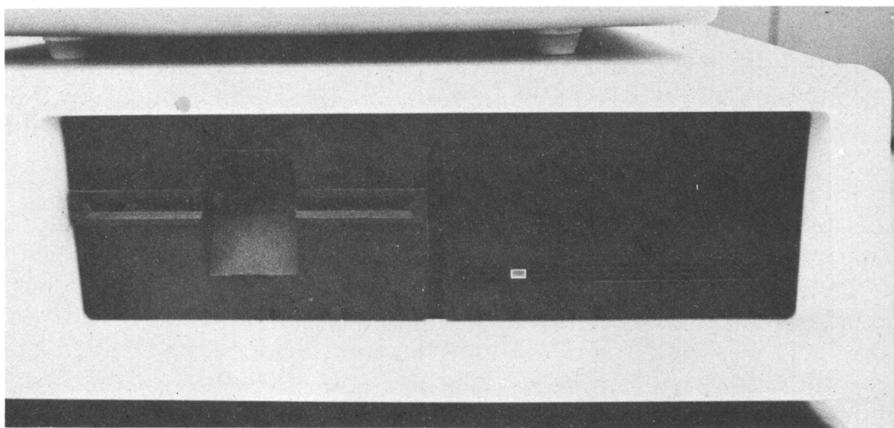


**Figura 1.1** L'unità di sistema del PC/XT IBM

La scheda di sistema, nella versione standard, comprende cinque connettori, chiamati *System Expansion Slots*, o porte di espansione del sistema (vedi Figura 1.2), che consentono l'inserimento di speciali schede opzionali; queste aumentano le capacità funzionali del PC e vi permettono così di adeguarlo alle vostre necessità specifiche. L'XT è dotato di otto di queste porte di espansione.

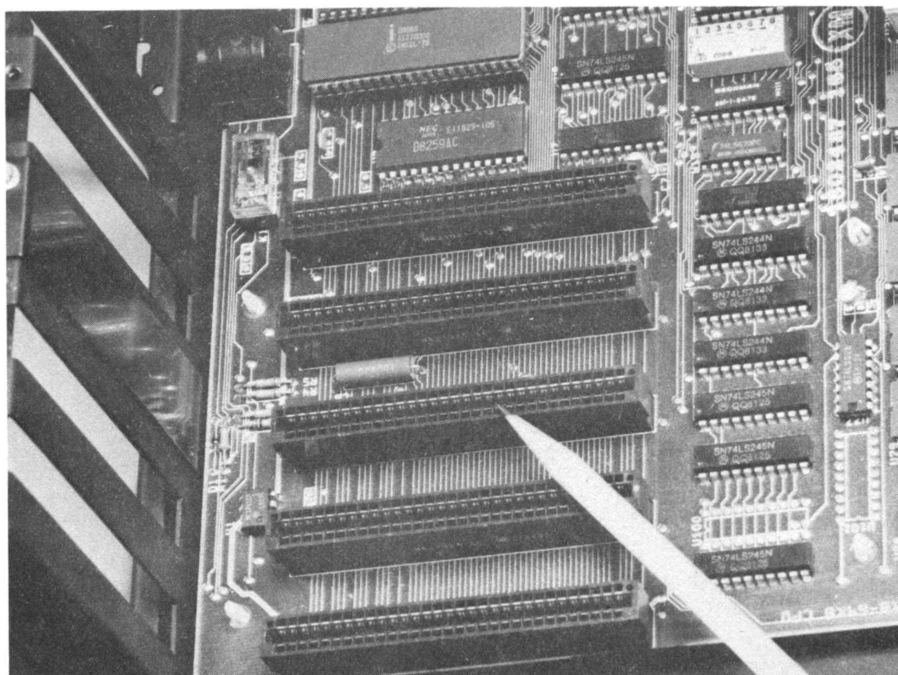


**Figura 1.2** Porte di espansione sulla scheda di sistema

## LA MEMORIA

La memoria del PC (vedi Figura 1.3), usata per immagazzinare dati e programmi, si divide in due categorie fondamentali: memoria di tipo ROM e memoria di tipo RAM. La ROM (*Read Only Memory*, cioè memoria a sola lettura) serve per dati e programmi che risiedono stabilmente nel PC, come il Cassette BASIC che viene fornito insieme al PC; la caratteristica principale della ROM è che il suo contenuto non viene perso quando si spegne il PC. La RAM (*Random Access Memory*, cioè memoria ad accesso diretto) è utilizzata per immagazzinare la maggior parte dei dati e dei programmi che vengono coinvolti nell'esecuzione; diversamente dalla ROM, il suo contenuto si perde quando il computer viene spento.
L'unità di misura della memoria, intesa sia in senso logico che fisico, è il *byte*: ad ogni byte corrisponde un valore che può essere diversamente interpretato, a seconda dell'operazione che viene eseguita. Un byte di memoria può rappresentare un numero, o un carattere, o comunque una singola informazione. La scheda di sistema ha normalmente una capacità

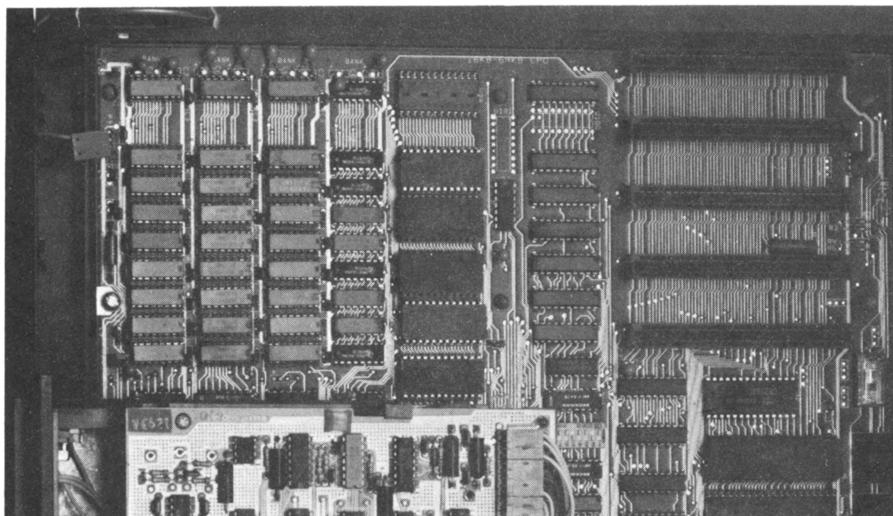


**Figura 1.3** Il microprocessore 8088 e la memoria sulla scheda di sistema

di 40 kilobyte, o KB, di ROM (1 kilobyte=1024 byte). Per quanto concerne la memoria RAM sono disponibili diverse versioni fino a 640KB, il massimo accessibile direttamente. È possibile aggiungere schede di espansione di memoria fino a 8MB, che possono essere usate solo come RAM Disk o memoria paginata.

## LA TASTIERA

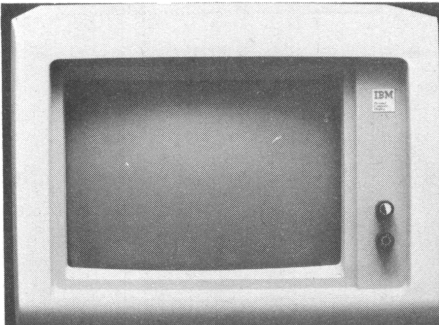
Come potete vedere in Figura 1.4, la tastiera del PC è molto simile a quella di una normale macchina da scrivere, ma porta anche dei tasti addizionali che vengono utilizzati per espletare speciali compiti: l'uso di questi tasti verrà ampiamente illustrato nel capitolo seguente.

## LO SCHERMO

Al vostro PC potete collegare quattro diversi tipi di schermo: il monitor monocromatico dell'IBM, un monitor in bianco e nero, un monitor a colori (come il monitor a colori IBM) o un normale televisore. Nella Figura 1.5 sono riportati i monitor IBM.




**Figura 1.4** La tastiera del PC



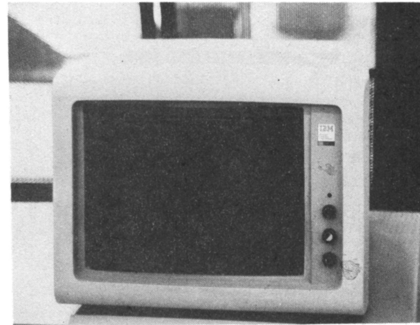



**Figura 1.5** Il monitor monocromatico e il monitor a colori IBM

In funzione del tipo di video usato, varierà il dispositivo necessario per il collegamento con il PC: per un monitor monocromatico IBM è indispensabile l'apposita scheda (*Monochrome Display and Parallel Printer Adapter*); un monitor a colori o in bianco e nero necessita invece della scheda Colore/Grafica (*Color/Graphics Monitor Adapter*). Naturalmente entrambi gli adattatori possono essere installati contemporaneamente. La scelta del dispositivo di visualizzazione dipende dall'uso primario che deve avere: per programmi che visualizzano solamente lettere e numeri è indicato il monitor monocromatico IBM; se invece intendete sfruttare colore e grafica per disegnare figure e diagrammi, sarebbe preferibile dotare il vostro PC della scheda Colore/Grafica. Il tipo di schermo meno costoso

che potete abbinare al vostro PC per mezzo di quest'ultimo adattatore è un normale televisore, ma purtroppo le differenze tra lo standard USA e quello europeo ne impediscono il collegamento. Per una buona qualità dell'immagine dovete usare un monitor RGB, con ingressi separati per i tre colori fondamentali: verde, rosso e blu. Il monitor a colori IBM è un monitor RGB ad alta risoluzione creato su misura per il collegamento con il PC.

## DISPOSITIVI DI MEMORIA DI MASSA

I dispositivi destinati a svolgere mansioni di memoria di massa permettono all'utente di conservare permanentemente dati e programmi. La velocità e la semplicità con cui si accede alle informazioni, le due principali qualità richieste ad una memoria di massa, dipendono naturalmente dal tipo di dispositivo usato.

### Drive per floppy disk

Il PC viene solitamente fornito già dotato di un drive per floppy disk a doppia faccia; inoltre solitamente si acquista anche un secondo drive da sistemare nell'apposito alloggiamento, come mostrato in Figura 1.6. Un disco a doppia faccia può contenere al più 360KB di informazioni: ovviamente un disco a faccia singola può contenere al più 180KB.
Per usare il drive bisogna installare in una porta di espansione del PC l'interfaccia per il drive, detta *Diskette Drive Adapter*: questa scheda di interfaccia permette al PC di controllare fino a 4 drive.

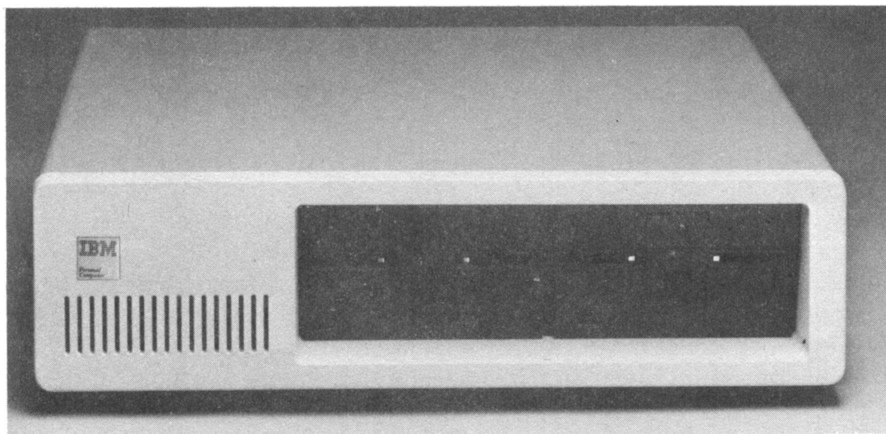



**Figura 1.6** Due drive montati nell'unità di sistema

### Il disco rigido

Il PC/XT è dotato di un disco rigido com una capacità di memoria di 10 o 20 megabyte, o MB (1 MB è all'incirca uguale a 1 milione di byte). Il disco rigido rappresenta un notevole miglioramento rispetto al floppy disk: non solo può contenere una quantità di informazioni notevolmente superiore, ma tutte le operazioni relative al disco rigido sono considerevolmente più veloci. Anche la versione standard del PC può essere corredata da un disco rigido con l'ausilio dell'unità di espansione IBM.

### Il registratore a cassette

Solo la versione standard del PC contiene un'interfaccia per il collegamento di un normale registratore a cassette, che può divenire in questo modo un dispositivo di memoria di massa; la versione XT non possiede una tale interfaccia, né può venirne dotata.

## 1.2 Periferiche standard

Il PC ha la possibilità di comunicare con varie altre categorie di dispositivi che stanno all'esterno dell'unità di sistema; per far ciò sono disponibili numerose schede, inseribili nelle porte di espansione del PC, che agiscono da interfaccia tra i vari dispositivi permettendo un adeguato collegamento e comunicazione tra essi.

### STAMPANTI IBM E INTERFACCIA PER STAMPANTE PARALLELA

La stampante a matrice di punti IBM, mostrata in Figura 1.7, può utilizzare un totale di 96 caratteri normali e di 64 caratteri grafici speciali con diversi stili di stampa e con la possibilità di scrivere 66, 80 o 132 caratteri per riga. La stampante grafica IBM, invece, è utile per altri scopi: consente una grande varietà nella dimensione dei caratteri e numerose realizzazioni grafiche.
Due sono i tipi di interfaccia disponibili per collegare stampante e PC: l'interfaccia per stampante parallela e la combinazione adattatore per monitor monocromatici e stampanti parallele, a cui abbiamo accennato poco sopra.

**ADATTATORE PER JOYSTICK**

Questa interfaccia, che potete vedere riprodotta in Figura 1.9, consente di collegare un joystick o un paddle al PC: questi dispositivi provvedono a rendere l'input più veloce che da tastiera e hanno un'ottima resa con quei programmi, come i giochi, che richiedono velocità.

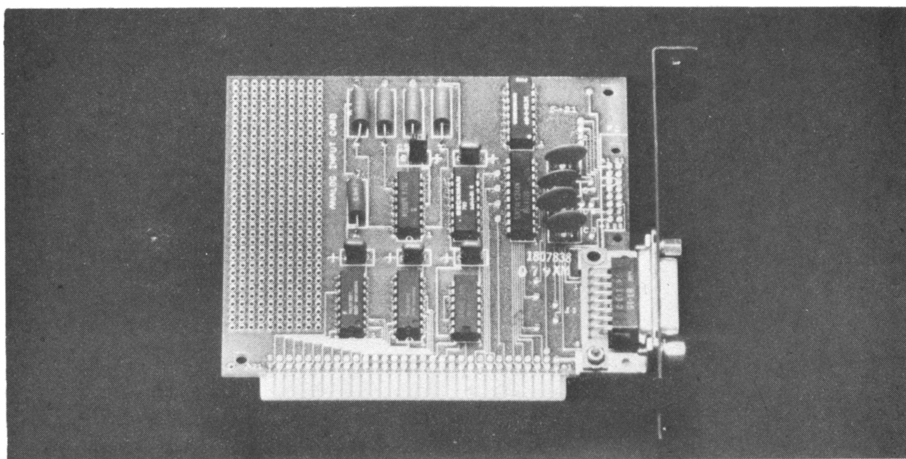


**Figura 1.9** L'adattatore per joystick e paddle

**UNITÀ DI ESPANSIONE**

Se volete aumentare le capacità del vostro PC, per esempio con l'aggiunta di nuove porte di espansione o del disco rigido IBM, dovete completare il vostro sistema con l'unità di espansione IBM (*Expansion Unit*); questa viene fornita completa dell'alimentatore, di un disco rigido da 10MB e di otto ulteriori porte di espansione per altrettante schede opzionali. L'unità, che ha le stesse dimensioni dell'unità di sistema e inoltre ha la possibilità di contenere un altro drive per dischi rigidi o floppy, può essere collegata sia al PC standard che all'XT.

# 1.3 Ulteriori opzioni

Per specializzare e personalizzare il vostro computer, cioè per renderlo in grado di svolgere compiti particolari, sono necessari altri dispositivi aggiuntivi (optional), prodotti non solo dall'IBM, ma anche da altre industrie.

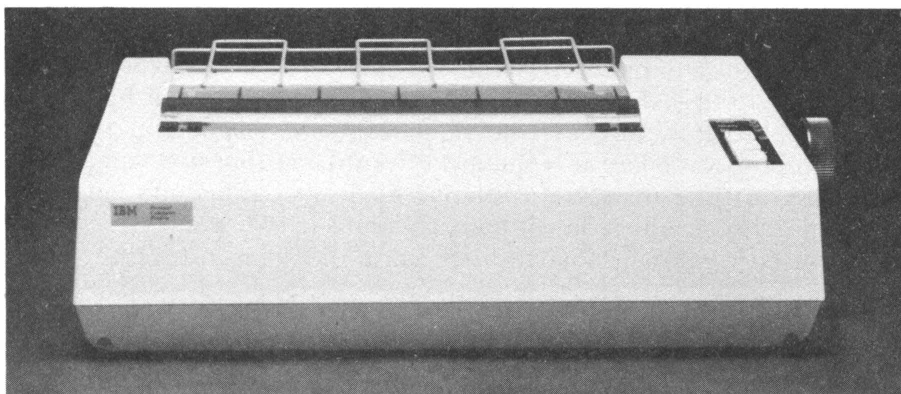


**Figura 1.7** La stampante a matrice IBM

## INTERFACCIA PER COMUNICAZIONI ASINCRONE

L'interfaccia per comunicazioni asincrone (*Asynchronous Communications Adapter*), mostrata in Figura 1.8, offre la possibilità di mettere in comunicazione il PC con diversi dispositivi per la trasmissione di dati. Questo tipo di interfaccia può essere usato per connettere il vostro PC con un altro PC, grazie ad un dispositivo predisposto per la trasmissione di dati su linea seriale asincrona RS232 o in *current loop* standard (come descritto nel Capitolo 11), oppure con un altro computer di diverso tipo — per esempio un database — su linea telefonica. La versione XT del PC viene fornita già dotata di questa utile interfaccia.

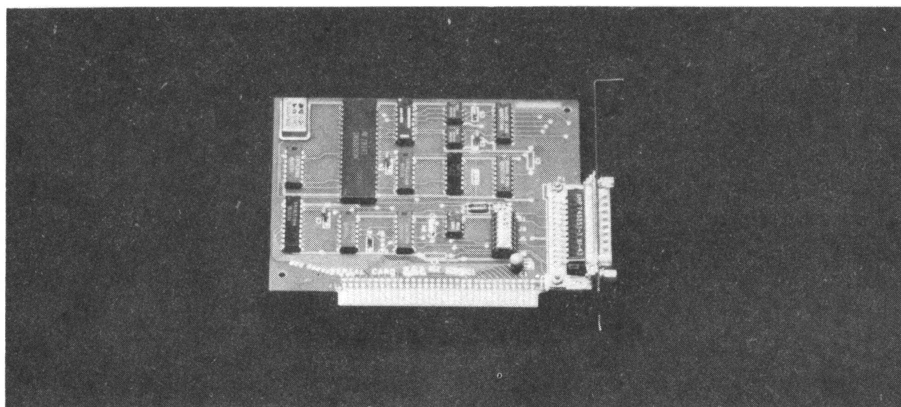


**Figura 1.8** L'interfaccia per comunicazioni asincrone

## SCHEDE DI MEMORIA

La quantità massima di memoria gestibile dal PC è di 640KB di RAM. Per raggiungere questa cifra, partendo dai 64KB di base sulla scheda di sistema sono disponibili un certo numero di schede di espansione con memorie RAM di capacità variabile da 64 a 512KB.
Nella Figura 1.10 è riprodotta una di queste schede di espansione di memoria. Questa particolare scheda può raggiungere i 256KB di RAM ed è dotata di altre tre opzioni: un orologio-calendario alimentato a batteria (perché il vostro PC continui a registrare lo scorrere del tempo anche quando è spento), una porta per dispositivi seriali e una porta parallela.

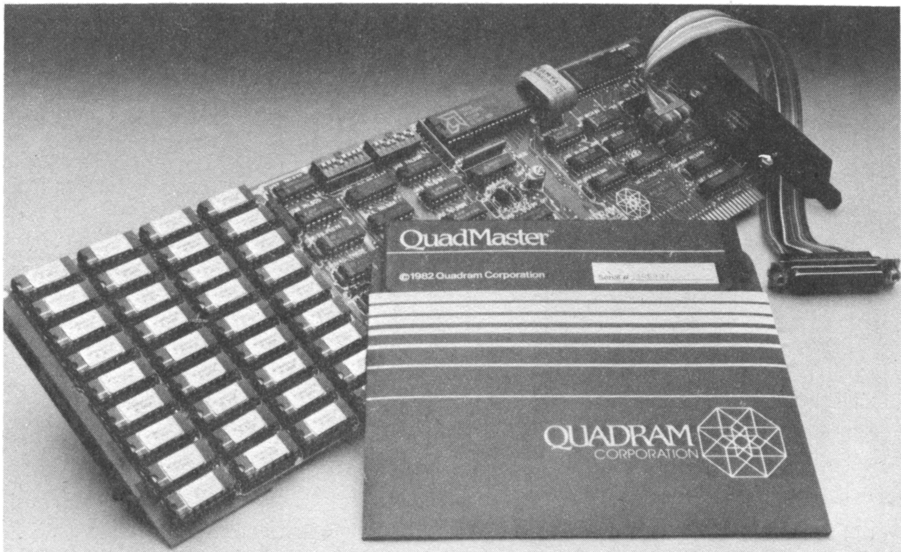



**Figura 1.10** Una scheda multifunzionale contenente 256KB di RAM, un orologio-calendario e interfacce parallele e seriali

## SCHEDE PER COMUNICAZIONE IN RETE

Questo particolare tipo di scheda permette di collegare tra loro numerosi PC in una disposizione chiamata rete locale (*network*). Tutti i sistemi collegati alla rete condividono dati e risorse di ogni tipo, ivi compresi dischi e stampanti.
Un gran numero di industrie produce reti standard: una delle più recenti e più diffuse è l'*Ethernet*. La scheda mostrata in Figura 1.11 vi permette appunto di collegare il vostro PC con altri PC in una rete *Ethernet*.

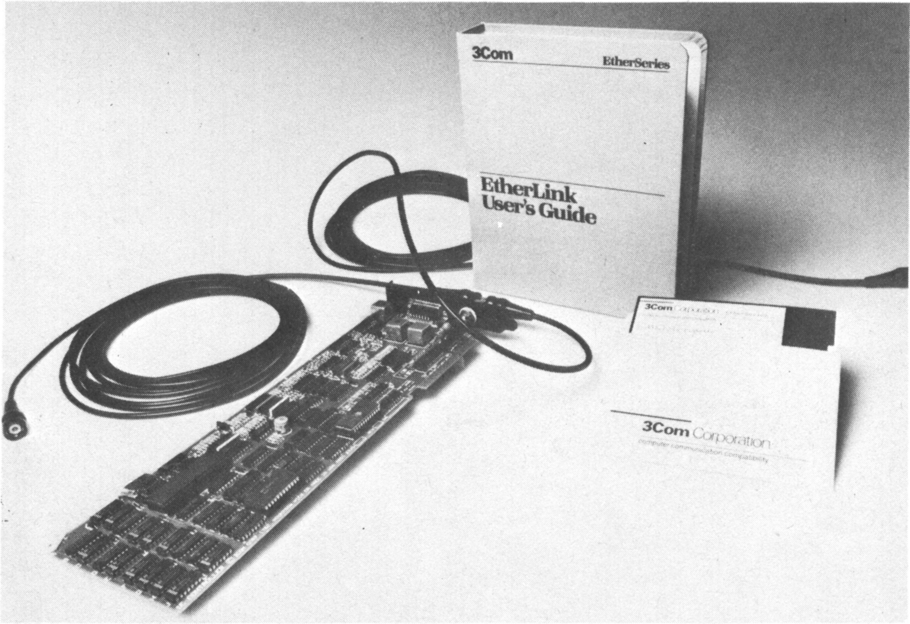



**Figura 1.11** Una scheda per comunicazione in rete

## CONTROLLORI DI I/O E SCHEDE PER ACQUISIZIONE DATI

Per controllare dispositivi esterni e raccogliere dati da diversi sensori posti nel "mondo esterno" il PC dev'essere dotato di controllori di I/O (input/output) e di schede per l'acquisizione dati. In Figura 1.12 vedete una scheda che svolge entrambe queste funzioni. Essa contiene un convertitore analogico-digitale che permette di tradurre misure provenienti da diverse fonti, fino ad un massimo di 16, in valori che il PC possa elaborare; inoltre questa particolare scheda è in grado di convertire impulsi digitali in segnali che possono controllare dispositivi analogici. Infine, tre porte parallele di output ad 8 bit vi consentono di controllare strumenti e dispositivi che richiedono un input in forma digitale.
Un impiego interessante di questo tipo di schede è il controllo e la gestione dei processi di laboratorio: un esperimento può essere seguito per mezzo degli input convertiti da analogici a digitali e può essere controllato per mezzo di output convertiti da digitali ad analogici attraverso le porte di output parallele.

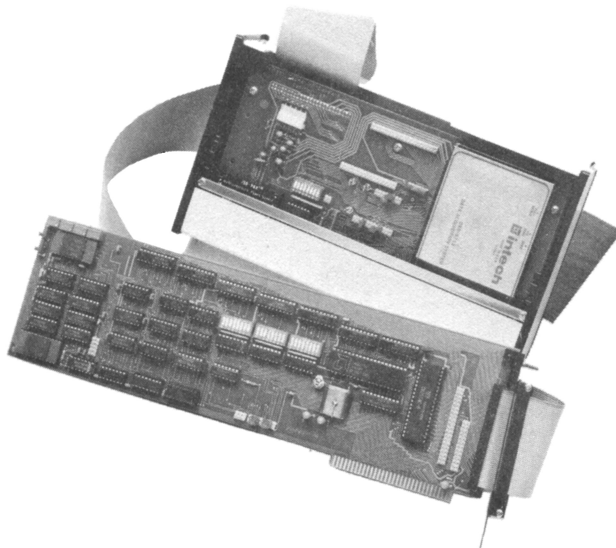


**Figura 1.12** Una scheda per acquisizione dati e controllo di I/O

## DISPOSITIVI PER INPUT GRAFICO

Un sistema molto elegante di fornire dati o istruzioni al PC è di disegnare a mano e di passare le informazioni per mezzo di un dispositivo per input grafico. L'informazione grafica può essere fornita sia in due che in tre dimensioni, a seconda del dispositivo utilizzato. Quello riportato in Figura 1.13 è una tavoletta per input grafico che permette di disegnare in tre dimensioni; l'unità è collegata al PC attraverso una porta seriale.

## BREADBOARD

Le *breadboard*, come quella mostrata in Figura 1.14, non sono altro che delle piastrine di bachelite per circuiti stampati su cui sono già state tracciate delle piste in rame secondo uno schema generale che permette di realizzare delle schede di interfaccia personalizzate. Queste schede contengono solitamente delle aree su cui montare i componenti, e le piste in rame vi consentono di accedere ai segnali provenienti dai canali di input/output del PC. Il progetto di tali schede è reso particolarmente semplice dal fatto che le specifiche complete delle caratteristiche elettriche del PC sono contenute nel manuale IBM *Technical Reference.*



**Figura 1.13** Una tavoletta grafica per input tridimensionale

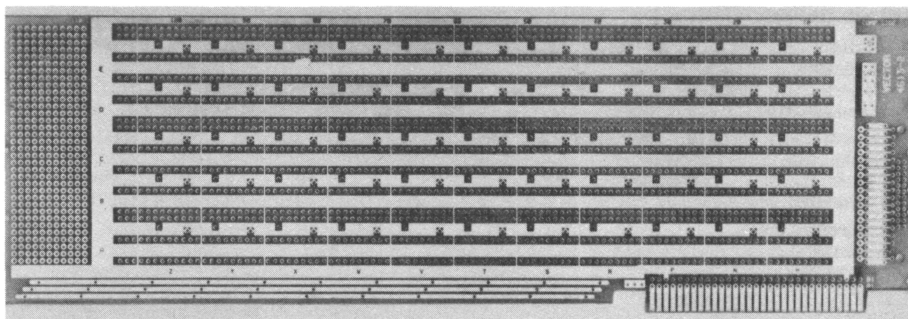


**Figura 1.14** Una breadboard

## 1.4 Il software per il PC

Le categorie in cui si può suddividere il software disponibile per il PC sono tre: sistemi operativi, linguaggi di programmazione e programmi applicativi.

### SISTEMI OPERATIVI

Un sistema operativo non è altro che un programma in grado di controllare l'hardware del PC: molti sono i sistemi operativi compatibili con il PC. In genere il tipo di programmi che intendete far eseguire dal vostro computer determinerà il tipo di sistema operativo più conveniente: un particolare programma spesso richiede che venga utilizzato, in modo corretto, un particolare sistema operativo.
Il sistema operativo più diffuso per il PC è il PC-DOS (chiamato comunemente DOS), una versione dell'MS-DOS della *Microsoft* prodotta dall'IBM: la versione 2.1 di questo sistema è quella a cui si farà sempre riferimento in questo libro.

### LINGUAGGI DI PROGRAMMAZIONE

Se pensate di scrivere da voi stessi i programmi da far eseguire al PC, avrete bisogno anche di un programma particolare: un interprete, un compilatore o un assemblatore a seconda del linguaggio che intendete usare.
Linguaggi ad alto livello come il BASIC, il Pascal, il FORTRAN, il COBOL e il FORTH vi permettono di usare nella programmazione frasi ed istruzioni che in una certa misura ricordano parole e simboli della lingua inglese; i linguaggi di tipo Assembler, invece, usano simboli e codici mnemonici di tre lettere per rappresentare istruzioni in linguaggio macchina e richiedono perciò una dettagliata conoscenza del suo funzionamento.
Tutti i PC vengono venduti con una versione del BASIC, detta *Cassette BASIC*, memorizzata su ROM (N.B.: anche se il PC/XT non ha una porta per il collegamento con le cassette, contiene anche il Cassette BASIC su ROM). I sistemi dotati di DOS vengono forniti anche di due versioni migliorate del BASIC: il *Disk BASIC* e l'*Advanced BASIC*. Gli altri linguaggi di programmazione devono essere acquistati separatamente.

## PROGRAMMI APPLICATIVI

I programmi applicativi, cioè i programmi creati per eseguire compiti specifici, possono essere acquistati e fatti eseguire direttamente, oppure possono essere scritti appositamente dall'utente in linguaggi come il BASIC o il Pascal.
I campi di applicazione sono realmente illimitati: qui descriviamo solo alcuni dei più comuni. Ricordate che un dato programma può sempre avere delle necessità specifiche per quanto riguarda la configurazione hardware del sistema: può essere richiesta una quantità minimale di memoria, o una particolare scheda opzionale. La diversa riuscita del programma può anche dipendere dai componenti del sistema: per esempio, molti tipi di programma vengono eseguiti molto più velocemente se hanno a disposizione maggiori quantità di memoria ad accesso diretto (RAM).

### 1. Word processing

I programmi di word processing sono tutti quei programmi che permettono una facile manipolazione dei testi, cioè consentono di creare un testo e di visualizzarlo sul video del PC. Esistono programmi in grado di controllare l'ortografia, di creare indici, di generare e conservare agende di indirizzi, ed in generale di eseguire tutti i compiti normalmente destinati ad una macchina da scrivere.

### 2. Fogli elettronici

I fogli elettronici (*Spreadsheets*) permettono di creare delle tabelle nelle quali il valore di ogni elemento può essere definito come costante o come funzione di altri elementi della tabella. Dopo che la tabella è stata realizzata, è possibile cambiare il valore di un qualsiasi elemento e tutti quelli che dipendono da questo valore verranno analogamente aggiornati. In questo modo potete definire un sistema di equazioni dipendenti che può venir usato per un gran numero di applicazioni, tra le quali, ad esempio, il bilancio preventivo e la previsione delle vendite.

### 3. Comunicazioni

Molti sono i programmi che permettono al PC di scambiare informazioni con altri computer; un programma di questo tipo può permettervi di accedere a parecchi servizi di banche dati per mezzo di un modem e di una linea telefonica. Inoltre i programmi di comunicazione sono in grado di

rendere il PC un "terminale intelligente" di un computer remoto, per esempio per trasferire file dal personal al mainframe e viceversa.

### 4. Contabilità

Il pacchetto di programmi di contabilità per il PC può seguire l'andamento finanziario dei vostri affari personali o di una piccola attività. Con programmi appropriati avrete la possibilità di registrare ogni operazione, tenere conti aggiornati, stampare fatture e bollette, e persino farvi i conteggi per le tasse, riempiendo il modello per la dichiarazione dei redditi.

### 5. Applicazioni scientifiche

Per coloro che hanno necessità di raccogliere grosse masse di dati, di controllare strumenti, di analizzare dati e creare modelli, fare previsioni e scrivere rapporti, sono disponibili programmi scientifici di applicazione in grado di svolgere una grande quantità dei compiti usuali per un laboratorio.

Capitolo

# Descrizione del PC-IBM

# 2

In questo capitolo vi mostreremo innanzitutto come installare ed attivare il PC. Per iniziare dovrete aver connesso almeno le seguenti parti:

— Unità di sistema con drive per floppy disk oppure, nell'XT, drive per dischi rigidi
— Tastiera
— Video ed interfaccia per il video

Per collegare tastiera e video all'unità di sistema e per installare correttamente tutte le schede d'interfaccia, dovete far riferimento alla documentazione tecnica fornita coi dispositivi.

## 2.1 Accensione del PC

Prima di accendere il PC dovete inserire il disco contenente il sistema operativo DOS nel drive A:. Estraete il disco dall'apposita busta che si trova nel manuale DOS, facendo attenzione a non piegarlo e a non toccarne la superficie non coperta dalla busta protettiva fissa.
Inserite il disco DOS nel drive A:, quello che si trova a sinistra (vedi Figura 2.1): aprite la chiusura del drive A:, tenete il disco tra pollice ed indice e con delicatezza infilatelo nella fessura fino a sentire un leggero clic; se inserito correttamente, il disco scivola facilmente fino a raggiungere la posizione definitiva, senza intoppi.

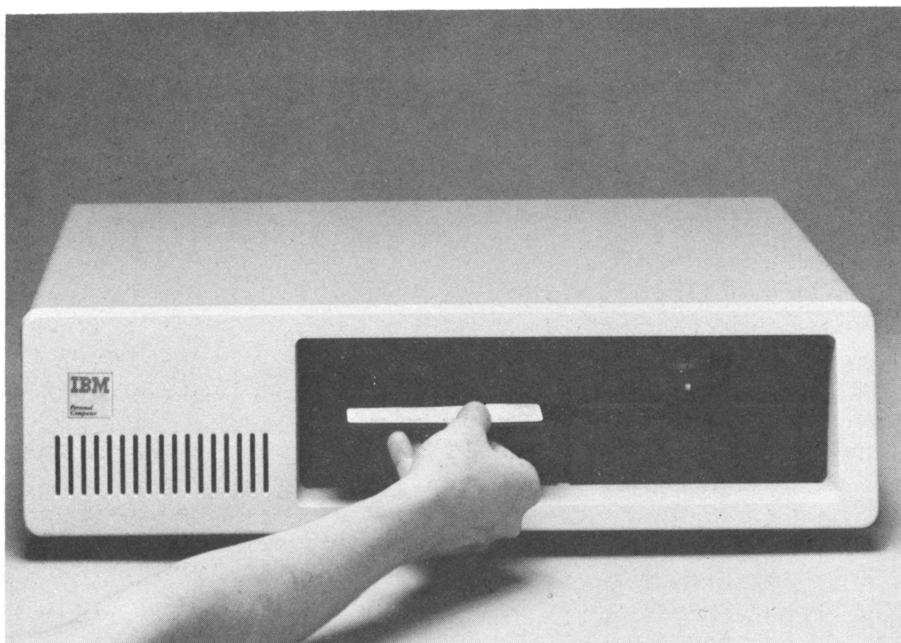



**Figura 2.1** Inserimento di un dischetto nel drive

Spingete ora con delicatezza lo sportello di blocco, fino a completa chiusura; se nell'operazione incontrate difficoltà, estraete il disco e riprovate: non forzate mai un disco ad entrare, piuttosto cercate di rimuovere la causa dell'ostruzione.
Controllate di nuovo che tutte le parti del sistema siano correttamente collegate tra loro e alla rete di alimentazione; accendete tutti i dispositivi periferici, come unità di espansione, monitor, e così via; infine accendete il PC azionando l'interruttore rosso che si trova sul lato destro dell'unità di sistema.
Immediatamente viene messo in azione il ventilatore, come potrete sentire dal caratteristico ronzio. Quando viene acceso, il PC esegue un certo numero di test di autodiagnostica, progettati appositamente per individuare eventuali problemi di hardware nel sistema. Questi test durano da pochi secondi fino a più di un minuto, a seconda delle dimensioni della memoria del vostro PC. Se possedete un XT, comparirà sullo schermo un segnale che indica l'esecuzione del test sulla memoria, mentre con la versione standard questa segnalazione vi verrà evidenziata solo al completamento dell'intera serie di test previsti.
Dopo un breve intervallo di tempo, un beep emesso dal vostro computer vi segnalerà la fine dei test di diagnostica e sullo schermo appariranno

tutti gli eventuali problemi evidenziati durante il controllo. Se vi imbattete in un problema, innanzitutto verificate la correttezza di tutti i collegamenti elettrici del sistema. Anche se molto rari, alcuni errori rilevati dal test possono essere sufficientemente gravi da impedire l'uso del PC se non vengono risolti.
Dopo il segnale acustico il PC automaticamente mette in azione il disco presente nel drive A: (la luce rossa posta sul fronte del drive si accende per indicarne l'attività): se il disco è proprio quello contenente il sistema DOS, questo verrà caricato.
Se non ci sono dischi nel drive, il PC standard, caricherà il Cassette BASIC; l'XT, invece, controllerà prima il disco rigido C: alla ricerca di una partizione contenente un valido sistema operativo (questo argomento verrà trattato nel Capitolo 3) e in caso affermativo lo caricherà, altrimenti anche l'XT utilizzerà il Cassette BASIC.
Come già accennato, il Cassette BASIC è la versione del BASIC memorizzata nella memoria ROM del PC. Poiché con il Cassette BASIC non potete usare i dischi né per leggere né per memorizzare dati e programmi, questa versione è di scarsa utilità e perciò non ne parleremo oltre. Se per errore aveste caricato il Cassette BASIC, potrete sostituirlo con il DOS semplicemente inserendo il disco con il sistema DOS nel drive A: e premendo contemporaneamente i tasti CTRL, ALT e DEL.

## INTRODUZIONE AL SISTEMA OPERATIVO

Il sistema operativo DOS fornisce all'utente i mezzi per comunicare con il PC, controllando quali programmi si possano far eseguire e quale parte dell'hardware venga usata e resa accessibile ai programmi stessi. Dopo aver avviato le procedure di inizializzazione del sistema, il DOS fa apparire sullo schermo il seguente messaggio:

```
A>keybit

A>wtdatim
Immettere la data odierna (GG-MM-AA): 01-01-1980
Modifica della data
```

A questo punto il DOS attende che voi inseriate la data del giorno corrente, che potete scrivere con il seguente formato:

```
8-02-1985
```

premendo infine il tasto ENTER; il PC vi risponderà:

```
Immettere l'ora: 00:00:24
Modifica dell'ora
```

Rispondete ora scrivendo l'ora corrente con un formato basato sulla suddivisione del giorno in 24 ore (da 0 a 23, dove 0 indica la mezzanotte), del tipo "hh:mm"; hh sta ad indicare l'ora ed mm i minuti. Per esempio, se sono le ore 7.30 del pomeriggio, dovrete scrivere:

```
19:30
```

e premere il tasto ENTER. Il PC vi risponderà infine:

```
IBM Personal Computer DOS Version  2.10

A>
```

Ora il sistema operativo DOS è diventato esecutivo. Si noti che quella appena illustrata, che è solamente una delle due vie per rendere operativo il DOS, viene chiamata "partenza a freddo" in quanto il suo avvio viene dato al momento stesso dell'accensione del computer. Se invece il computer è già stato acceso precedentemente, potete reinizializzare il sistema, con il metodo detto "partenza a caldo", inserendo il disco con il sistema DOS nel drive A: e premendo contemporaneamente i tasti CTRL, ALT e DEL.

Se possedete un XT o un PC standard dotato dell'unità di espansione IBM, potete predisporre il disco rigido, invece del floppy disk inserito nel drive A:, ad avviare il DOS. Per ulteriori dettagli fate riferimento a "Partizioni del disco rigido" nel paragrafo 3.5.

## CARICAMENTO DEL BASIC DALL'INTERNO DEL DOS

Dopo aver reso operativo il DOS potrebbe esservi utile caricare il Disk BASIC, una versione avanzata del BASIC; a questo fine inserite la parola:

```
A>BASIC
```

in risposta al prompt del DOS. (Se fate un errore di battitura correggete usando il tasto BACKSPACE).

NOTA: ogni qualvolta vogliate "informare" il sistema operativo DOS di aver terminato la scrittura del comando, battete il tasto ENTER.

Il PC ora cerca sul disco del DOS inserito nel drive A: il programma BA-

SIC.COM e lo carica in memoria; al completamento dell'operazione sullo schermo appare la seguente scritta:

```
The IBM Personal Computer Basic
Version D2.10 Copyright IBM Corp. 1981, 1982, 1983
61763 Bytes free

Ok
```

A questo punto siete in grado di caricare e far eseguire programmi scritti in BASIC, di scrivere i vostri programmi o di eseguire calcoli immediati (in modo diretto): tutti questi argomenti verranno trattati nei capitoli successivi. Per tornare al DOS dopo aver caricato il BASIC dovete dare il comando:

```
SYSTEM
```

Riappare immediatamente il prompt A> caratteristico del DOS ad indicare che il computer è ritornato al sistema operativo.

## 2.2 Uso della tastiera

La tastiera è lo strumento più usato per comunicare con il PC e poiché ad essa dedicherete gran parte del vostro tempo, è bene iniziare subito a esaminarla.

### DISPOSIZIONE DELLA TASTIERA

Osservate la tastiera, come riportata in Figura 2.2: dedicate alcuni minuti a prendere confidenza con le principali aree: i tasti alfabetici, numerici, di punteggiatura sono disposti in maniera simile a quelli di una normale macchina da scrivere; i dieci tasti funzione si trovano all'estrema sinistra, mentre i dieci che formano il tastierino numerico sono posti sulla destra. Identificate infine tre tasti con funzioni particolari che userete molto frequentemente: ENTER, ALT e CTRL.

#### Tastiera base

Questa è la parte che utilizzerete maggiormente per l'inserimento di dati o in genere di informazioni: funziona esattamente come una normale

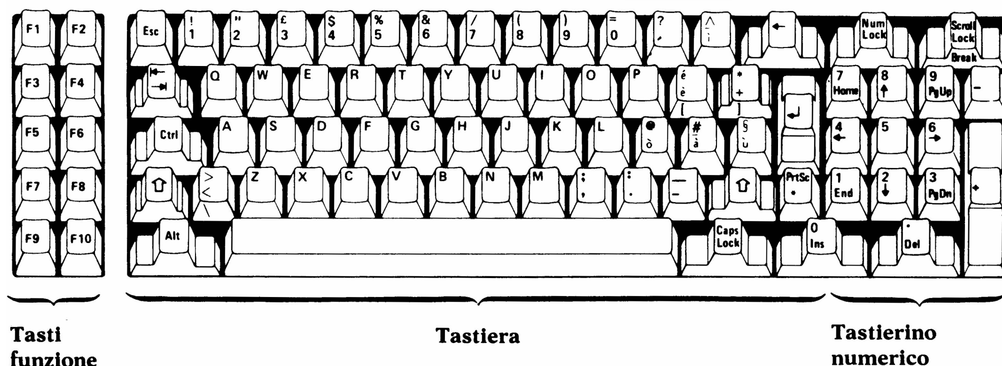



**Figura 2.2** La tastiera del PC

macchina da scrivere; per ottenere le maiuscole o i caratteri riportati nella parte alta dei tasti dovete utilizzare il tasto SHIFT.

## Tasti funzione

I tasti funzione sono disposti su due file, alla sinistra della tastiera e numerati da F1 a F10; il loro scopo è di rendere più semplice l'invio di alcuni comandi usati frequentemente. La loro azione è differente a seconda del programma corrente: in BASIC ognuno di questi tasti equivale alla battitura di un completo comando BASIC (i comandi corrispondenti ad ogni tasto sono visualizzati nella parte inferiore dello schermo).

## Tastierino numerico

Il tastierino numerico è composto dai tasti che si trovano sulla destra della tastiera; essi hanno due funzioni principali: in primo luogo possono essere usati per battere numeri quando il programma ne richiede una grande quantità in input. Questa disposizione è più comoda e razionale e consente una scrittura più veloce dei numeri; inoltre questi tasti possono essere usati, nell'ambito del BASIC, per muovere il cursore: le quattro diverse direzioni sono indicate dalle frecce che compaiono sui tasti numerici.
La diversa funzione del tastierino numerico, sempre in BASIC, viene impostata per mezzo del tasto NUM LOCK: premendolo una volta il tastierino funziona per inserire numeri, premendolo di nuovo il tastierino permette di controllare la posizione del cursore.

### Tasto ENTER

Il tasto ENTER (↵), una volta premuto, comunica al PC che avete appena terminato di scrivere la linea che volete far eseguire. Nella maggior parte dei casi quello che scrivete non verrà preso in considerazione dal PC finché non viene premuto il tasto ENTER.

### Tasti ALT e CTRL

Il tasto ALT (*Alternate*) e il tasto CTRL (*Control*) vengono usati insieme ad altri tasti per inviare parole o comandi speciali al PC. Per esempio, nell'ambito del BASIC, tenendo premuto il tasto ALT e battendo il tasto P si ottiene la visualizzazione del comando PRINT.

## FUNZIONI DI CONTROLLO DEL SISTEMA

Tre funzioni peculiari permettono di esercitare un controllo sull'azione del PC. Esse sono: l'inizializzazione del sistema (*system reset*), l'interruzione (*break*), e la sospensione (*pause*).

### Inizializzazione del sistema

Questa è la funzione di cui abbiamo già parlato, che viene attivata premendo contemporaneamente i tasti CTRL, ALT e DEL. Una reinizializzazione del sistema fa sì che il PC si ritrovi nelle condizioni iniziali, come se fosse stato appena acceso. Quando inviate un comando di reset, viene interrotto il programma in esecuzione e vengono persi tutti i dati che non siano stati precedentemente salvati su disco. A questo punto il PC cerca un disco contenente un sistema operativo (DOS, CP/M 86 o altri), prima nel drive A: e poi nel drive C:. Se trova un disco del tipo cercato, il sistema operativo viene caricato e mandato in esecuzione; se invece nel drive A: non vi sono dischi o non esiste un drive C: o, infine, il drive C: non ha sezioni contenenti un sistema operativo, viene avviato il Cassette BASIC.

### Interruzione

La funzione di interruzione, che si attiva premendo nello stesso momento i tasti CTRL e SCROLL LOCK/BREAK, agisce sul programma in esecuzione in quel momento: se, per esempio, avete fatto eseguire un programma partendo dal BASIC, il comando fa terminare l'esecuzione del programma e fa riapparire il prompt caratteristico del BASIC.

### Sospensione

Premendo contemporaneamente i tasti CTRL e NUM LOCK si richiama la funzione di sospensione: questa interrompe temporaneamente l'esecuzione del programma corrente. Per far riprendere l'esecuzione basta premere un tasto qualsiasi. Questa funzione risulta molto utile quando un programma o un comando invia sullo schermo una quantità di informazioni superiore a quella che potete leggere in una volta sola: in questo caso sospendete l'esecuzione del programma finché non abbiate letto tutto ciò che vi interessa e quindi fate riprendere la visualizzazione premendo un qualsiasi tasto.

## SEMPLICI CORREZIONI

Se commettete un errore mentre state scrivendo, potete correggere la linea, prima di aver premuto il tasto ENTER, in molti modi diversi che dipendono dal tipo di programma in esecuzione. Per il momento esaminiamo solo due dei metodi che consentono di correggere una linea di scrittura.

### Tasto BACKSPACE

Nell'ambito del BASIC o del DOS potete usare il tasto BACKSPACE (←) che muove il cursore di una posizione verso sinistra cancellando contemporaneamente il carattere che vi si trovava.

### Tasto ESC

L'uso del tasto ESC (*Escape*) equivale a riportare il cursore all'indietro, cancellando un'intera riga. In BASIC questa operazione semplicemente cancella la riga su cui è posizionato il cursore; in DOS la linea in questione viene segnata con il simbolo \ ed il cursore si sposta alla riga successiva, dove potete riscrivere la linea corretta.

## COME SCRIVERE UN BREVE PROGRAMMA IN BASIC

Vi mostriamo ora come scrivere un breve programma in BASIC. Innanzitutto caricate il BASIC, come abbiamo spiegato prima, nel seguente modo: in ambiente DOS battete il comando BASIC e premete il tasto ENTER; apparirà il prompt caratteristico del BASIC, Ok.

Ora scrivete sul PC le due linee seguenti, copiandole esattamente come sono scritte; se vi accorgete di un errore dopo aver battuto ENTER, semplicemente riscrivete di nuovo l'intera linea. Per ora non preoccupatevi del significato che queste istruzioni possono avere in BASIC: ne parleremo ampiamente nei prossimi due capitoli.

```
10 INPUT "Inserite il vostro nome e cognome --",X$
20 PRINT "Ciao, " X$ ", come va?"
```

Ora battete il seguente comando BASIC per eseguire il programma:

```
RUN
```

Il PC immediatamente visualizza:

```
Inserite il vostro nome e cognome --
```

Proseguite rispondendo alla richiesta, per esempio:

```
Inserite il vostro nome e cognome -- Mario Colombo
```

Quando premete ENTER, ad indicare di aver completato la risposta, il PC replicherà:

```
Ciao, Mario Colombo, come va?
Ok
```

L'apparire del prompt Ok indica che l'esecuzione del programma è terminata.
Avete caricato ed eseguito il vostro primo programma in BASIC. Questo era solo un esempio di poca importanza: nei capitoli seguenti esamineremo in dettaglio il problema del caricamento e dell'esecuzione dei programmi in BASIC. Per il momento torniamo al DOS battendo il comando:

```
SYSTEM
```

## 2.3 Uso dei dischi

Esaminiamo ora gli elementi fondamentali dell'uso dei dischi: con il termine dischi ci riferiamo sia ai floppy disk che ai dischi rigidi, tranne quando specificato esplicitamente.

Nell'unità di sistema del PC possono essere alloggiati uno o due disk drive; quello a sinistra è sempre un drive per floppy disk ed è detto drive A:; quello a destra è un drive opzionale per floppy disk nella versione standard del PC, ed è chiamato drive B:, mentre nell'XT è un drive per dischi rigidi, chiamato drive C:. Un'unità di espansione, collegata sia al PC standard che all'XT permette di alloggiare nell'unità di sistema due drive per floppy disk (A: e B:) e due dischi rigidi (C: e D:) nell'unità di espansione. Se il vostro sistema è dotato di un solo drive, questo viene utilizzato sia come drive A: che come drive B:, quando necessario. Un programma od un comando che debbano indirizzarsi ad entrambi, considerano l'unico drive come due unità separate, alternando l'invio di istruzioni ed avvisando l'utente, al momento opportuno, di inserire un determinato disco.

## USO DEI FLOPPY DISK

I floppy disk sono una delle parti più delicate del vostro sistema: sono molto sensibili alla polvere e ad ogni tipo di sporcizia e risentono dei campi magnetici. Se però avete cura di non dimenticare alcune piccole precauzioni potete evitare la maggior parte di questi problemi.
Non toccate la superficie del disco e abbiate cura di non piegarlo mai; tenetelo lontano da ogni sorgente di campi magnetici (televisore, telefono, motori, ...) e non esponetelo mai al calore o alla luce del sole.
Riponete sempre i dischi nella custodia dopo averli usati: questo vi aiuterà anche a tenerli in ordine e a saperli sempre identificare. Scrivete le etichette prima di applicarle al disco o utilizzate con delicatezza un pennarello e mai penne o matite la cui punta rigida potrebbe danneggiare il disco.
Infine, non inserite e non estraete mai il disco dal drive quando la luce rossa è accesa: questo potrebbe distruggere parzialmente i dati in esso contenuti.

## COME VISUALIZZARE IL CONTENUTO DEL DISCO

Le informazioni sono memorizzate sul disco in unità chiamate *file*. Vogliamo ora sapere quali file si trovino sul disco contenente il sistema operativo DOS.
Innanzitutto assicuriamoci che il DOS sia caricato: se già non lo avete fatto, inserite il disco nel drive A: e accendete il PC oppure operate una reinizializzazione del sistema, inserendo poi data e ora correnti in risposta alle richieste del DOS.

Ora che il DOS è sicuramente esecutivo battete la seguente parola:

```
DIR
```

a seguito del prompt caratteristico del DOS.
Il PC legge il catalogo del disco inserito nel drive A: e visualizza un elenco di tutti i file in esso memorizzati. Useremo alcuni di questi file nella successiva trattazione; maggiori dettagli saranno forniti nel Capitolo 3.

## COME COPIARE I DISCHI

È buona abitudine fare copie di sicurezza (*backup*) dei vostri dischi, nell'eventualità di qualche danno all'originale.
Per prima cosa procuratevi un disco nuovo e rimuovete l'eventuale copertura della tacca di protezione contro la sovrascrittura (vedi Figura 2.3).
Battete ora il comando:

```
DISKCOPY
```

e seguite le istruzioni che via via appaiono sullo schermo per copiare tutte le informazioni contenute nel disco sorgente (in questo caso il disco contenente il DOS) nel disco destinazione (in questo caso il disco nuovo).
Ora coprite la tacca di protezione del disco-copia ed usate normalmente questo; riponete l'originale in un posto sicuro.

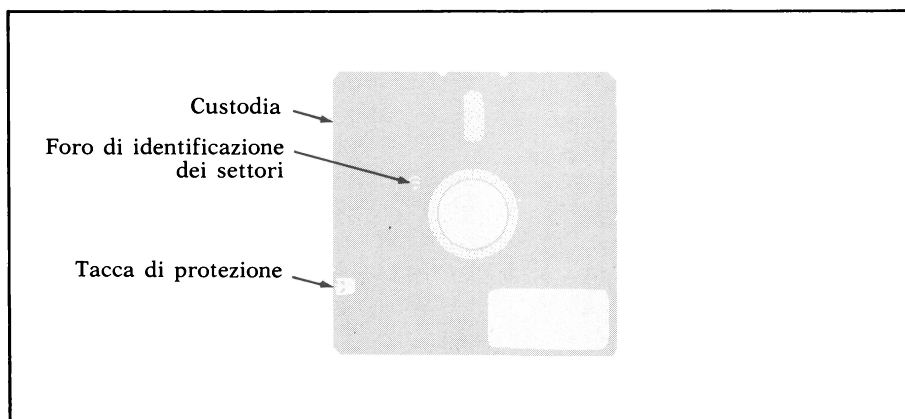



**Figura 2.3** La tacca di protezione contro la sovrascrittura su un dischetto

### COME CARICARE ED ESEGUIRE PROGRAMMI MEMORIZZATI SU DISCO

Per spiegare come caricare ed eseguire i programmi che sono memorizzati su un disco, dapprima manderemo in esecuzione il programma Advanced BASIC, seguito dal programma chiamato SAMPLES, che si trova sul disco "Programmi complementari" venduto insieme al DOS 2.10.
Dopo aver reso operativo il DOS battete il comando:

```
BASICA
```

e il DOS andrà alla ricerca, nel disco inserito nel drive A:, del file chiamato BASICA.COM caricandolo nel PC ed eseguendolo; a questo punto vi apparirà la videata tipica dell'Advanced BASIC.
Togliete ora dal drive A: il disco DOS ed inserite il disco "Programmi complementari"; battendo il seguente comando:

```
LOAD "SAMPLES"
```

verrà caricato all'interno della memoria del PC il programma SAMPLES.BAS e premendo il tasto funzione F2 o battendo il comando RUN lo si manderà in esecuzione.
Una volta terminato di usare il programma SAMPLES, basterà premere il tasto ESC per ritornare al BASIC, oppure battere il comando SYSTEM, dopo aver sostituito nel drive A: il disco "Programmi complementari" con il disco DOS, per ritornare al DOS.

## 2.4 Uso della stampante

Se possedete una stampante, avete a disposizione diversi modi di creare delle copie su carta di quanto appare sullo schermo. Quando si desidera usare la stampante, prima di tutto bisogna fare attenzione che sia connessa al PC come mostrato nella Figura 2.4 e che la carta sia stata inserita correttamente nel rullo di trascinamento. La vostra stampante sarà in grado di ricevere dati dal PC dopo l'accensione e l'attivazione della linea di collegamento, indicata dall'accensione della luce corrispondente al tasto ON LINE.
Quando volete far avanzare la carta dovete premere di nuovo ON LINE ed usare i tasti FF (*form feed*) per far avanzare di una pagina o LF (*line feed*) per farla avanzare di una riga, oppure spegnere la stampante e ruotare manualmente il rullo di avanzamento. Non cercate mai di far tornare indietro la carta ruotando il rullo in senso contrario, perché potreste dan-

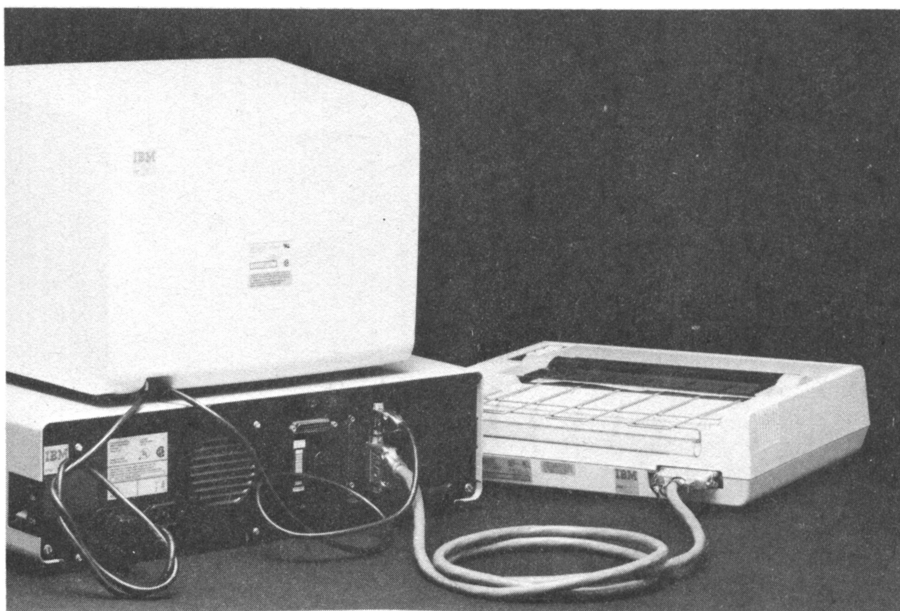


**Figura 2.4** La stampante IBM collegata al PC

neggiare il meccanismo. Quando sarete nuovamente pronti per la stampa, ricordatevi di premere ON LINE.

## COPIA DELLO SCHERMO

Per fare una copia su carta di quello che appare sullo schermo in un dato momento sarà sufficiente che premiate contemporaneamente i tasti SHIFT e PRTSC; il PC copierà sulla stampante, linea per linea, tutta la videata.

## COPIA DIRETTA

Premendo i tasti CTRL e PRTSC quando state lavorando con il DOS o in BASIC, tutto ciò che scriverete con la tastiera e che otterrete in risposta dal PC verrà immediatamente riportato sulla stampante. Per tornare alle condizioni normali dovrete premere nuovamente i tasti CTRL e PRTSC.

Capitolo

# Il sistema operativo 3

Diversi sistemi operativi trovano impiego sul PC; in questo capitolo vi spiegheremo qual è la funzione del sistema operativo standard, la versione 2.10 del PC-DOS, conosciuto anche sotto i nomi di MS-DOS, Microsoft DOS o semplicemente DOS.
Dopo la lettura di questo capitolo vi sentirete quasi padroni del DOS: i suoi comandi principali verranno ampiamente esemplificati; per completezza, inoltre, nell'Appendice D, al termine del libro, sono riportati tutti i comandi in ordine alfabetico.

## 3.1 Le funzioni del DOS

Il compito principale del DOS è di permettere ai programmi che voi utilizzate, di comunicare con l'hardware del PC: a questo scopo sono predisposti i programmi che costituiscono il DOS, che ci consentono di controllare il fluire delle informazioni tra i vari componenti, come video, tastiera, scheda di interfaccia, memoria, drive.
L'azione del DOS è spesso invisibile: l'utente non si accorge, nella maggior parte dei casi, di quando il DOS entra in funzione. Supponiamo, per esempio, che un programma di word processing sia giunto al momento in cui il testo dev'essere stampato: per questo il programma userà la routine del sistema operativo che consente il trasferimento dei caratteri dalla memoria alla stampante. Sarà la routine stessa a provvedere ad ogni problema riguardante la temporizzazione e la formattazione dei dati da trasferire, ma l'intero processo si svolgerà "dietro le quinte" e voi non vi ac-

corgerete che il sistema operativo è entrato in funzione.
Il DOS viene invece utilizzato esplicitamente quando volete compiere operazioni come la copia di un disco, l'esame del suo contenuto o la cancellazione di informazioni dallo stesso: per tutti questi scopi dovete inviare comandi propri del DOS direttamente dalla tastiera.

## 3.2 I file DOS

Un file è un insieme di informazioni ordinate, diversamente accessibile a seconda della sua ubicazione fisica nel PC.
I file DOS possono essere trasferiti tra tutti i dispositivi elencati nella Tabella 3.1: la maggior parte dei file che userete si troverà su disco, ma potete trasferirli attraverso tutti questi dispositivi per mezzo del DOS. Ad esempio il comando COPY, illustrato nel Capitolo 2 per produrre copie di sicurezza dei dischi, può essere anche usato per trasferire file da un dispositivo DOS ad un altro.

**Tabella 3.1** Nomi dei dispositivi DOS

| Dispositivo | Nome |
|---|---|
| Drive per floppy | A: o B: |
| Drive per dischi rigidi | C: o D: |
| Tastiera e video | CON: |
| Prima porta per interfaccia asincrona | AUX: o COM1: |
| Seconda porta per interfaccia asincrona | COM2: |
| Prima stampante parallela | PRN: o LPT1: |
| Seconda o terza stampante parallela | LPT2: o LPT3: |
| Dispositivo non esistente o "dummy" | NUL: |

### ORGANIZZAZIONE DEI FILE DOS

In genere un PC con un drive ha un catalogo (*directory*) con una struttura molto semplice: tutti i file di un dato dischetto sono elencati in un unico catalogo simile ad un elenco telefonico in cui sono riportati tutti gli abitanti di una determinata località in ordine alfabetico insieme ai rispettivi indirizzi e numeri di telefono. Il catalogo del disco contiene, oltre ad altre informazioni che ora non ci interessano, il nome di ogni file ed un indirizzo nascosto che consente al DOS di identificare sul disco, il singolo file.

Se il sistema, invece, è dotato di dischi rigidi, come nel caso della versione XT, un singolo catalogo diventerebbe poco utile. A causa della notevole capacità di memoria è possibile immagazzinare letteralmente centinaia di file su un disco rigido; è bene perciò che l'utente possa organizzare il catalogo, e di conseguenza i file memorizzati, secondo le sue necessità. Il catalogo con struttura "ad albero" del DOS è la soluzione a questo problema.
Mentre il catologo semplice può essere paragonato all'elenco telefonico, quello con struttura ad albero è più simile alle *Pagine Gialle*, che suddividono un gran numero di imprese nelle categorie commerciali di appartenenza: può esserci, per esempio, una voce che riguarda le automobili; se vi occorre un servizio o un prodotto per automobili, esaminate quella voce. Naturalmente possono esserci delle suddivisioni interne: rivenditori, ricambi, riparazioni, e così via. Attraverso una struttura gerarchica di sezioni e sottosezioni potete arrivare velocemente all'indirizzo giusto per il vostro problema. (Provate ad ottenere lo stesso risultato usando il normale elenco telefonico!).
La struttura ad albero del DOS crea una simile struttura gerarchica di "sottocataloghi" (*subdirectory*) per ordinare i vostri file. Quando format-

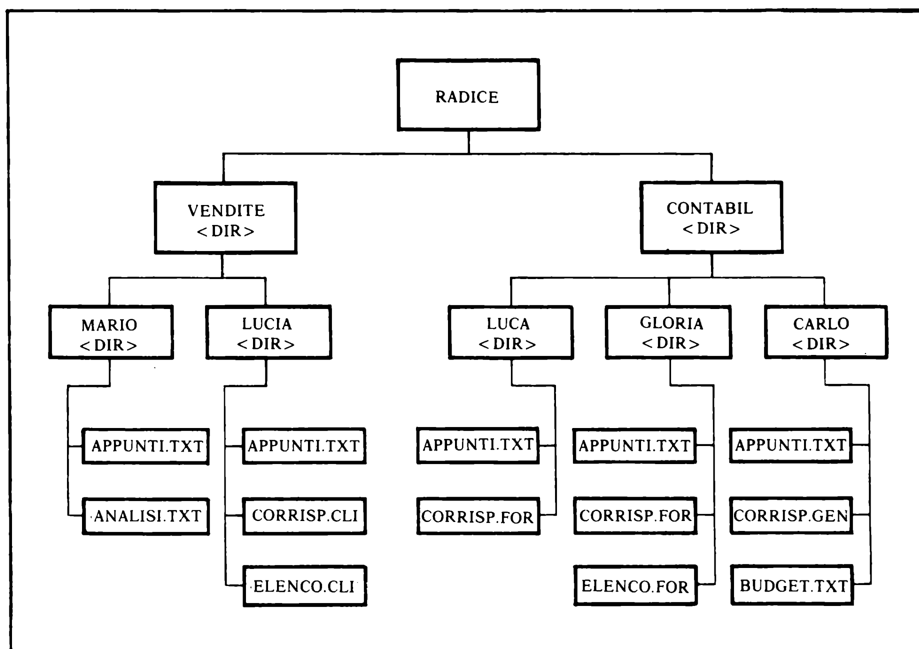



**Figura 3.1** Struttura ad albero

tate un disco (comando FORMAT) viene creato un catalogo, chiamato *directory di sistema* o *radice*. Usando speciali comandi DOS (MKDIR, RMDIR e CHDIR) potete creare, cancellare o comunque manipolare dei subdirectory per questo disco; ognuno di questi può contenere file DOS o subdirectory di se stesso. Una simile struttura è mostrata in Figura 3.1. (Per capire da cosa prende il nome la struttura ad albero immaginate la figura capovolta: dalla radice si dipartono verso l'alto dei rami — file o subdirectory — formando una specie di albero).
Nel nostro esempio compaiono due prime suddivisioni della radice (VENDITE e CONTABIL), ciascuna delle quali corrisponde ad un dipartimento della società ed ha, a sua volta, un subdirectory per ogni impiegato di quel settore che contiene tutti i file relativi allo stesso (ad esempio, appunti, corrispondenza, ecc.). Potete facilmente constatare che alcuni file portano lo stesso nome, ma questo non causa confusione in quanto i file appartengono a subdirectory diversi.
La successione dei subdirectory che dalla radice portano ad un file specifico è detta *cammino* per quel file; la radice ed ogni eventuale ramificazione sono indicate in un cammino dal simbolo \ (*backslash*): dall'illustrazione precedente si ricava che il cammino per il file APPUNTI.TXT di Mario è dato da:

```
\VENDITE\MARIO
```

Qualora, dopo aver creato un catalogo, esaminiate l'elenco dei file ivi contenuti utilizzando il comando DIR, vi accorgerete che esistono due tipi di nomi con una caratteristica distintiva, quelli che contengono un punto e quelli che ne contengono due: sono file particolari che riportano informazioni sul catalogo.
Il subdirectory corrente è rappresentato da un punto seguito dall'identificatore <DIR>; il catalogo da cui dipende il subdirectory corrente, detto "padre", è rappresentato da due punti. Sempre per rifarci all'esempio precedente: il catalogo radice è "padre" sia di VENDITE che di CONTABIL, mentre VENDITE è "padre" di MARIO e di LUCIA.
I due punti all'inizio di un cammino significano: "inizia il cammino con il padre del subdirectory corrente".
Il comando CHDIR è quello che vi permette di muovervi all'interno di differenti subdirectory; infatti il DOS è in grado di identificare e marcare come directory "corrente" quello verso cui intendete muovervi, in modo che tutti i comandi inviati e tutte le manipolazioni di file vengano effettuati sugli elementi di questo, a meno che non segnaliate di voler fare un ulteriore cambiamento.
Si osservi che, al momento della formattazione del disco, viene creato solo il directory radice a struttura indifferenziata, che rimane l'unico in attività fintantoché non desideriate crearne altri.

Come si è già accennato prima, tre sono i comandi DOS che permettono di manipolare i cataloghi ad albero. Si faccia sempre attenzione a specificare nel comando la lettera identificativa del drive cui ci si rivolge, altrimenti verrà considerato per default il drive A:.
I comandi sono:

— Creazione di un nuovo subdirectory

MKDIR [*d:*] *cammino*

oppure

MD [*d:*] *cammino*

(MD, come MKDIR, è l'abbreviazione di *Make Directory*).

— Cambio del directory corrente

CHDIR [*d:*] [*cammino*]

oppure

CD [*d:*] [*cammino*]

(CD, come CHDIR, è l'abbreviazione di *Change Directory*).

Si osservi che, a differenza del precedente comando, in quest'ultimo l'argomento *cammino* è racchiuso tra parentesi, ad indicare che il suo inserimento è facoltativo: se non viene specificato, sul video apparirà il cammino del directory corrente.

— Cancellazione del directory

RMDIR [*d:*] *cammino*

oppure

RD [*d:*] *cammino*

(RD, come RMDIR, è l'abbreviazione di *Remove Directory*).

Quando si utilizza quest'ultimo comando, bisogna ricordarsi di svuotare il directory di file e subdirectory prima di cancellarlo; a questo punto gli unici elementi visualizzati nella lista dovrebbero essere quelli con uno e con due punti.

Naturalmente potete costruire la gerarchia interna del catalogo ad albero nella maniera che preferite ed adattarla poi secondo le vostre scelte, oppure potete ignorare completamente questo tipo di approccio ed usare il directory indifferenziato sia per il floppy disk che per il disco rigido.

## NOMI DEI FILE DOS

Ogni file nel DOS viene specificato tramite una serie di quattro "specificazioni": identificatore del drive, cammino, nome del file, estensione, come mostrato di seguito:

[*d:*] [*cammino*] *nomefile* [*.est*]

dove le parentesi quadrate stanno ad indicare che si tratta di parti opzionali nella frase identificativa del file. Quello che segue è un esempio di quanto appena detto

```
C:\VENDITE\MARIO\APPUNTI.TXT
```

Lo specificatore di drive *d:* può essere A: o B: per i floppy disk A e B, oppure C: o D: per i drive C e D per dischi rigidi, e quando viene omesso, al suo posto viene usato il valore di default, che è quello specificato dal carattere prompt del DOS, nel nostro caso A>.
L'indicazione del *cammino*, che è opzionale, permette di raggiungere il subdirectory della struttura ad albero in cui è contenuto il file desiderato; naturalmente, se questa indicazione viene trascurata, il directory cui il sistema si riferirà è il corrente.
Delle ultime due parole, cioè il nome del file e la sua estensione, che permettono di identificare inequivocabilmente un file nel catalogo di un disco, particolarmente importante è la prima, dato che ovviamente in un disco possono essere contenuti parecchi file.
In generale, comunque, è possibile riferirsi anche a file che non si trovino su disco, utilizzando semplicemente, nella frase di specificazione del file, il nome del dispositivo verso cui indirizzarsi seguito dai due punti; i nomi DOS dei vari dispositivi sono riportati nella Tabella 3.1.
Per quanto riguarda il nome del file e l'estensione, le principali caratteristiche sono le seguenti:

— Il nome del file deve essere lungo da uno a otto caratteri
— L'estensione deve essere lunga da uno a tre caratteri
— Il nome del file e l'estensione possono contenere i seguenti caratteri:

lettere dalla A alla Z

numeri da 0 a 9

$ & # @ ! % ' () -

Qui di seguito proponiamo alcuni nomi di file validi secondo le regole prima esposte:

```
A:PROCESS.BAS
B:Tasse%.FTN
CON:
PUNT#12.MUS
LPT1:
```

Ed ora alcuni non validi seguiti dalla spiegazione:

| | |
|---|---|
| BE BOP.BRD | Lo spazio bianco non è un carattere valido |
| BIBLIOTECA.LIB | È troppo lungo |
| "2NAVIG.2B" | Il carattere " non è valido |

## CARATTERI JOLLY PER NOMI DI FILE

I cosiddetti caratteri jolly per nomi di file, vi consentono di specificare in maniera generica un file o un gruppo di file. Infatti, se avete un gruppo di file i cui nomi iniziano tutti con lo stesso carattere, potete usare il carattere jolly per avere una lista di tutti questi file in una volta sola.
Il DOS accetta due soli caratteri con questa funzione, purché siano inseriti nella stringa di specificazione del file or ora vista: sono i caratteri ? e * e la loro interpretazione è spiegata qui di seguito.
Quando nella specificazione di un file compare il carattere ?, il DOS fa sì che la sua posizione venga progressivamente occupata da tutte le possibili lettere dell'alfabeto. In questo modo se ad esempio sul disco contenuto nel drive A: si trovano molti file il cui nome è formato da una singola lettera seguita dalla stringa NOVEL, tutti aventi come estensione .TXT; per ottenerne la lista basterà dare il comando DIR nel seguente modo:

```
A>DIR ?NOVEL.TXT
```

ed il PC vi risponderà:

```
ANOVEL.TXT
BNOVEL.TXT
CNOVEL.TXT
```

Analogamente avviene con il carattere *, con la differenza che la posizio-

ne dell'asterisco nella stringa di specificazione può essere occupata da più di un carattere; infatti se si desidera la lista di tutti i file che iniziano con una singola lettera seguita dalla stringa NOVEL, indipendentemente dal tipo di estensione, basterà dare il comando:

```
A>DIR ?NOVEL.*
```

ottenendo in risposta:

```
ANOVEL.TXT
ANOVEL.BAK
BNOVEL.TXT
BNOVEL.BAK
CNOVEL.TXT
```

## FILE DOS SU DISCO

Sono ora opportune alcune considerazioni sui due tipi di drive che possono accompagnare il PC: quelli per floppy disk e quelli per dischi rigidi. Essi differiscono soprattutto per la quantità di dati memorizzabili e per la velocità con cui si può accedere ai dati; in entrambi i casi si dimostrano più efficienti i dischi rigidi. Va ricordato che al contrario dei floppy, i dischi rigidi non possono essere rimossi e sostituiti dall'utente.

### File per floppy disk

I floppy disk sono in grado di contenere dati su una sola o su entrambe le facce del disco, a seconda delle possibilità del drive che avete collegato al PC. I dischi adatti per i drive IBM contengono, su ogni faccia, una serie di 40 tracce concentriche, molto simili ai solchi di un normale disco musicale: ogni traccia è suddivisa in otto o nove settori, lunghi, a loro volta, 512 byte.
Delle 40 tracce, solo 39 sono disponibili per dati ed informazioni, mentre la prima viene riservata dal DOS per una mappa del disco e per tutti i dati relativi al disco che è utile sapere. Da tutto ciò risulta che il massimo di informazioni che potete memorizzare su un disco a singola faccia è dato da:

$$39 \text{ tracce} \times 9 \text{ settori/traccia} \times 512 \text{ byte/settore} =$$
$$= 179712 \text{ byte (o caratteri)}$$

Naturalmente i dischi a doppia faccia avranno una capacità doppia, cioè all'incirca di 360000 byte.
Per quanto riguarda il numero dei file su disco, un disco a singola faccia ne può contenere al massimo 64, mentre un disco a doppia faccia 112.
Nell'uso dei floppy disk non vi deve importare di conoscere alla perfezione come i dati siano effettivamente disposti nel disco, perché il DOS si prende carico di ogni problema al vostro posto, mentre dovete porre attenzione alla quantità di informazioni che inviate a ciascun disco, sia in termini di byte o caratteri (180000 o 360000, rispettivamente per dischi a singola e a doppia faccia) che in termini di numero massimo di file (64 o 112).

**File per dischi rigidi**

I dischi rigidi IBM hanno una capacità di più di 10 o 20 milioni di byte o caratteri.
Poiché il disco rigido, diversamente dal floppy disk, non può essere rimosso dal sistema, (come indicato dal termine *fixed*, che lo identifica in inglese) è indispensabile qualche metodo per ottenere copie dei file in esso memorizzati (le copie di sicurezza di cui abbiamo già fatto menzione).
Sono due comandi del DOS, BACKUP e RESTORE, che permettono questa periodica copia del disco rigido.
I dischi rigidi, inoltre, per mezzo del comando FDISK possono essere suddivisi in aree distinte l'una dall'altra che contengono diversi sistemi operativi (DOS, CP/M-86 e simili).

## 3.3 Inizializzazione ed uso del DOS

Come visto nel Capitolo 2, sono due i metodi che permettono di caricare il DOS:

1. Accendere il PC dopo aver inserito un dischetto del sistema operativo nel drive A:, oppure dopo essersi assicurati che un'area selezionata del disco rigido contiene il DOS.
2. Operare una reinizializzazione del sistema (premendo contemporaneamente i tasti CTRL, ALT e DEL) con i dischi come nella precedente situazione.

Dopo che il DOS è stato così caricato, il PC vi richiede data e ora, come già visto, e infine visualizza il prompt:

```
A>
```

Questo prompt ha diversi significati: innanzitutto vi informa che il DOS è in attesa che voi inseriate un comando specifico del DOS stesso; inoltre vi ricorda qual è il drive di default o drive corrente. (Nell'esempio precedente è il drive A:). Questo è il drive a cui vengono inviati tutti i comandi nel caso in cui venga omessa la specificazione del drive.
È possibile d'altra parte cambiare il drive corrente, con un semplice comando:

| | |
|---|---|
| `A> B:`<br>`B>` | Cambia il drive corrente<br>da A: a B: |
| `B> A:`<br>`A>` | Cambia il drive corrente<br>da B: ad A: |

## USO DEL DOS CON UN UNICO DRIVE

Anche se il vostro sistema è dotato di un unico drive per floppy disk, potete ugualmente indirizzarvi sia al drive A: che al drive B:, in modo che dal punto di vista logico sia come avere due drive diversi, mentre fisicamente ne avete a disposizione uno solo. Se il DOS deve effettuare qualche operazione che richiede l'uso di due floppy disk, vi avvisa dopo aver terminato con il primo: a questo punto voi lo estraete ed al suo posto inserite il secondo. Il DOS, in questo modo, utilizzerà un disco per volta, avvertendovi sempre quando è il momento di sostituire i dischi.
Supponiamo di voler copiare un file da un disco ad un altro, sempre utilizzando il sistema con un solo drive. Innanzitutto inseriamo nel drive A: il disco contenente il file originale; per dare avvio all'operazione di copia battiamo il comando:

```
A> COPY SEGRE.EQ B:
```

Il DOS carica quanto più può del file SEGRE.EQ dal disco inserito e poi vi avvisa di sostituire al disco iniziale (quello del drive logico A:) il disco destinazione (quello del drive logico B:) con la seguente frase:

```
Insert diskette for drive B:
and strike any key when ready
```

Nel drive A: è ora inserito il secondo disco: questa alternanza dei dischi continua finché il file non è completamente copiato. Tutti i comandi per la cui esecuzione sono necessari due dischi avranno un funzionamento simile nei sistemi con un solo drive.

## USO DEI COMANDI DOS

I comandi del DOS hanno una forma specifica: sono una combinazione di una parola chiave che indica la funzione del comando e di uno o due specificatori. Maiuscole e minuscole possono essere usate indifferentemente: tutti i parametri del comando verranno automaticamente convertiti in lettere maiuscole.
Entrambi i seguenti comandi, perciò, avranno l'effetto di copiare il file ONEMORET.IME dal disco rigido del drive C: nel nuovo file PLYTAGN.SAM sul disco nel drive A:

```
COPY C:ONEMORET.IME A:PLYTAGN.SAM
copy C:oneMOREt.ime a:plytagn.sam
```

### Delimitatori di comandi

Le differenti parti di un comando DOS devono essere opportunamente separate per mezzo di uno o più simboli presi tra i seguenti: spazi vuoti, virgole, punto e virgola, segno di uguale o tabulazioni. Ecco alcuni esempi di comandi validi:

```
TYPE,SHORTSTO.TXT
COPY VECCHIOF.ILE;B:NUOVOF.BAK
RENAME      UNNOME.BAS      UNALTRO.BAS
DIR         C:\VENDITE\*.*
```

Notate che in ogni caso un segno (:) deve seguire lo specificatore di dispositivi ed un punto deve precedere l'estensione; ancora, in un cammino i subdirectory sono indicati dal segno \.

### Correzione degli errori nei comandi DOS

Spesso vi verranno utili due funzioni di editing del DOS per correggere gli errori di battitura dei comandi: queste sono rappresentate dal tasto BACKSPACE e dal tasto ESC. Il primo viene usato per cancellare un carattere per volta e in particolare quello che si trova alla sinistra del cursore: dopo aver cancellato il carattere, il cursore si sposta verso sinistra e vi permette così di inserire nella posizione corretta il carattere sostitutivo. Se la linea corrente è affetta da troppi errori, potrebbe risultare lungo sistemarla con l'ausilio del tasto BACKSPACE: è più utile, in questo caso, cancellare l'intera linea per mezzo del tasto ESC. La linea non viene cancella-

ta dallo schermo, ma semplicemente le viene aggiunto il simbolo (\) al termine ed il cursore si sposta nella linea successiva per consentire la nuova battitura di tutta la linea.

### SEQUENZE DI CONTROLLO DEL DOS

Un piccolo numero di tasti svolge un ruolo particolare nel sistema operativo DOS: alcuni di questi sono già stati menzionati nel Capitolo 2. Potete vedere tutte queste funzioni elencate nella Tabella 3.2, ricordando però che alcune speciali funzioni di editing, corrispondenti ad altri tasti, saranno discusse nel Capitolo 12.

**Tabella 3.2** Sequenze di controllo del DOS

| Combinazione di tasti | Funzione |
|---|---|
| CTRL BREAK | Interrompe l'esecuzione del comando DOS o del programma corrente. Ritorna al livello "comandi DOS" |
| CTRL NUM LOCK | Sospende temporaneamente l'esecuzione del comando DOS; per riprendere, premere un qualsiasi tasto |
| SHIFT PRTSC | Viene stampata la videata corrente |
| CTRL PRTSC | La stampante copia ogni linea di output nel momento in cui viene visualizzata sullo schermo |
| CTRL ENTER | Muove il cursore alla linea successiva senza mandare in esecuzione il comando |

### MESSAGGI D'ERRORE DOS

Ad ogni specifico comando o programma contenuto nel disco DOS sono abbinati dei messaggi d'errore: la maggior parte di questi viene visualizzata in seguito ad errori di battitura o di sintassi di un comando. L'elenco di questi messaggi è riportato nell'Appendice E.

## 3.4 Comandi DOS

Ogniqualvolta battete qualcosa in risposta ad un prompt del sistema operativo, il DOS si aspetta che voi abbiate inviato un comando DOS valido; questo può essere un comando *interno* o un comando *esterno*, a seconda

della locazione in cui risiede la routine corrispondente al comando.
Un comando DOS interno è compreso in quella parte di routine che il DOS carica in memoria: ogni volta che battete un comando interno, questo viene immediatamente eseguito.
I comandi DOS di tipo esterno sono memorizzati su disco e quando ne viene richiesto uno, il DOS va a ricercare il corrispondente programma sul disco, lo carica in memoria ed infine lo esegue. Molti sono i comandi DOS standard che risiedono sul disco del DOS e sono trattati come comandi esterni.
I file che contengono i comandi esterni sono caratterizzati dal fatto che l'estensione del nome del file può essere .EXE o .COM; per utilizzare un comando esterno, però, è sufficiente scrivere il nome del file senza estensione. Abbiamo già visto che, per caricare l'Advanced BASIC, il comando è semplicemente:

```
BASICA
```

che fa sì che il DOS ricerchi il programma che porta lo stesso nome, lo carichi in memoria e lo esegua.
Lo scopo di questa distinzione tra comandi esterni e comandi interni è quello di risparmiare memoria senza rallentare eccessivamente l'esecuzione. I comandi più usati sono inseriti come comandi interni, e così possono essere eseguiti più velocemente; gli altri, al contrario, sono memorizzati sul disco DOS come comandi esterni, in modo da lasciare in memoria spazio sufficiente, per esempio, per i vostri programmi.
Poiché, a volte, i comandi interni vengono chiamati "comandi residenti" possiamo riferirci ai comandi esterni anche come "comandi transitori".
Nella sintassi di molti dei comandi DOS rientra la frase di specificazione del file, che abbiamo esaminato a pag. 56.
Ricordiamo qui la sintassi della specificazione: [*d:*] [*cammino*] *nomefile* [*.est*] in cui le parentesi quadre indicano l'opzionalità del parametro in esse racchiuso.
Il parametro *d:* specifica il drive: se omesso, viene scelto il drive di default. Il *cammino* indica il percorso per raggiungere un catalogo attraverso una struttura ad albero; se non indicato, viene utilizzato il catalogo corrente.
Il *nomefile* può essere facoltativo solo nel caso in cui la specificazione stessa sia posta tra parentesi quadre nella sintassi del comando, altrimenti deve sempre essere indicato. *est.*, opzionale, indica naturalmente l'estensione del nome del file.

## 3.5 Comandi DOS per la manipolazione dei dischi

La prima operazione per cui vi sarà utile il DOS è la cura dei vostri dischi: per questo iniziamo questo paragrafo con l'esposizione dei comandi DOS che permettono di preparare e sistemare i dischi da usare con il PC.

### COME FORMATTARE UN DISCO: IL COMANDO FORMAT

Prima di poter memorizzare, con l'ausilio del DOS, dati ed informazioni di ogni tipo su un dischetto nuovo, è indispensabile prepararlo usando una procedura chiamata formattazione.
Il comando FORMAT, utilizzato a questo scopo, ha molte funzioni: innanzitutto scrive sul disco alcune informazioni che permettono di suddividerlo in tracce e settori. Il modo in cui il disco viene formattato è specifico del DOS: un disco formattato su un computer diverso dal PC o anche semplicemente con un diverso sistema operativo, molto probabilmente non funzionerà con il DOS.
È essenziale, a questo punto, che comprendiate come il comando FORMAT distrugga ogni informazione precedentemente memorizzata su quel disco: non potete perciò usare questo comando per convertire i dati di un disco non compatibile con il PC in modo da poterlo utilizzare. Inoltre, se il disco contiene dei file validi per il DOS, dopo la formattazione essi saranno irrimediabilmente persi. Ponete perciò molta attenzione nel controllare il contenuto del disco prima di formattarlo.
Attenzione: se il vostro sistema è un PC/XT, dovete ulteriormente aumentare le precauzioni; un incauto uso del comando FORMAT potrebbe distruggere immediatamente il contenuto del vostro disco rigido!
Dopo aver completato l'organizzazione del disco, il programma di formattazione controlla eventuali aree difettose nel disco e infine scrive le informazioni che permettono al DOS di immagazzinare e recuperare programmi e file di dati: il directory del disco (il catalogo dei file memorizzati) e la tabella di allocazione dei file (una sorta di indice interno che informa il DOS su come lo spazio del disco è utilizzato e a quale file è assegnato).
A richiesta dell'utente il comando FORMAT permette di aggiungere al disco informazioni specifiche del DOS, come copie dei file del sistema DOS.
Ecco qual è la sintassi del comando FORMAT:

FORMAT [*d:*] [/S] [/1] [/8] [/V] [/B]

Se viene specificato il drive, verrà formattato il disco nel drive indicato, altrimenti quello nel drive di default. (Attenzione: controllate che il disco

da formattare sia inserito nel drive corretto).
L'opzione /S permette di inserire sul nuovo disco, dopo la formattazione, una copia del DOS, oltre ai due file "nascosti" IBMBIO.COM e IBMDOS.COM e al file COMMAND.COM (I file "nascosti" sono così chiamati perché il loro nome non appare nel catalogo del disco quando viene listato).
I parametri /1 e /8 hanno validità solo per i floppy disk; normalmente il comando FORMAT prepara le due facce del disco dividendole in nove settori per ogni traccia. Nel caso che voi vogliate usarne una sola, o che il drive permetta l'uso solo di dischi ad una sola faccia, utilizzate l'opzione /1, in modo da creare dischi a faccia singola. (Nota: un drive a doppia faccia può usare un disco a singola faccia, ma non viceversa). Il parametro /8, invece, permette di suddividere le tracce in otto settori invece che in nove. Le prime versioni del DOS (1.10 e 1.00) possono usare solo dischi con otto settori per traccia.
Se vi interessasse identificare il disco che state formattando con un nome specifico (chiamato *volume label*) richiedete l'opzione /V: dopo la formattazione il DOS vi richiederà la volume label, che può contenere fino a 11 caratteri. Molti comandi DOS, tra i quali DIR e VOL visualizzeranno la volume label se presente sul disco utilizzato.
Il parametro /B, quando specificato, consente la produzione di dischi con otto settori per traccia e con uno spazio riservato per i moduli del sistema DOS, IBMBIO.COM e IBMDOS.COM. Questi file nascosti e il file COMMAND.COM non vengono copiati sul nuovo disco, ma possono essere aggiunti successivamente per mezzo del comando SYS. L'opzione /B non ha effetto sui dischi rigidi.
Al termine di tutte le operazioni il programma di formattazione vi fornisce tutte le informazioni che possono esservi utili: lo spazio totale del disco, l'eventuale spazio riconosciuto difettoso, lo spazio assegnato ai file di sistema eventualmente copiati e lo spazio rimasto libero.
I dischi rigidi sono già formattati all'atto della vendita del computer, ma è possibile formattarli di nuovo (sempre con il comando FORMAT) anche se unicamente all'interno di una sezione (partizione) contenente il DOS. Per maggiori dettagli leggete il paragrafo "Partizioni del disco rigido: il comando FDISK" a pag. 68.
Riassumendo, per creare un disco a doppia faccia, con 9 settori per traccia, con una copia del DOS ed una volume label usate il comando:

```
A>FORMAT B:/s/v
```

## COME CREARE UN DISCO DI SISTEMA: IL COMANDO SYS

Se avete usato le opzioni /S o /B nella formattazione del dischetto, avete riservato su di esso delle locazioni speciali destinate a contenere i file di sistema nascosti: questi due file, IBMBIO.COM e IBMDOS.COM devono trovarsi nelle locazioni riservate in ogni disco di sistema valido.
Il comando SYS, di cui riportiamo la sintassi, copia la versione del DOS contenuta nel disco di default nel disco specificato:

SYS *d:*

(Come appena detto, il parametro *d:* indica il drive in cui si trova il disco destinazione dei file di sistema). Questo comando è utile quando acquistate un disco di programmi applicativi supportati dal DOS in cui non ci sia il sistema operativo, ma solo lo spazio riservato, in modo che possiate inserirvi la versione più aggiornata del DOS.
Naturalmente questo comando può essere utilizzato solo con dischi precedentemente formattati con la specificazione dei parametri /S o /B. Potete anche aggiornare un disco che contenga una versione del DOS non recente: SYS sostituisce la versione originale del DOS con quella che si trova sul disco inserito nel drive corrente. Nell'esempio seguente sono riportati i comandi che permettono di copiare sul dischetto del drive A: i file di sistema presenti sul disco rigido C:

```
C>SYS A:
System transferred
C>
```

## COME COPIARE L'INTERO CONTENUTO DI UN DISCO: IL COMANDO DISKCOPY

Se volete riprodurre il contenuto di un dischetto, è molto utile il comando DISKCOPY, che ha la seguente sintassi:

DISKCOPY [*d1:*] [*d2:*] [/1]

dove il primo specificatore di drive identifica il drive sorgente (quello da cui si copia) e il secondo il drive destinazione (quello su cui si deve scrivere). Se tralasciate entrambi gli specificatori di drive, la copia avverrà tramite il solo drive di default e il PC vi avviserà ogniqualvolta sarà necessario alternare i due dischi. Il comando non è in grado di operare una copia dei dischi rigidi.

Il parametro /1 fa sì che venga copiata una sola faccia del disco, indipendentemente dal tipo di dischetto o di drive utilizzato; usualmente, invece, il comando DISKCOPY rileva il numero delle facce e dei settori del disco sorgente e li riproduce esattamente sul disco destinazione.
Inoltre, questo è l'unico comando DOS che agisce anche su dischi non formattati: se il disco destinazione non è stato precedentemente formattato, il comando lo formatterà prima dell'operazione di copia. Nell'esempio seguente vediamo la copia di un dischetto a singola faccia; fate attenzione alla richiesta di inserimento del disco nel drive A: che viene utilizzato come due diversi drive:

```
C>DISKCOPY A: B:/1

Insert source diskette in drive A:

Strike any key when ready

Copying 9 sectors per track, 1 side(s)

Insert target diskette in drive A:

Strike any key when ready

Copy complete

Copy another (Y/N)?N
C>
```

## COME CONFRONTARE DUE DISCHI: IL COMANDO DISKCOMP

Dopo aver copiato un disco, per controllare che l'operazione sia stata compiuta correttamente, potete confrontare il contenuto dei due dischi con l'ausilio del comando DISKCOMP (anche questo non ha effetto sui dischi rigidi). La sintassi, molto simile alla precedente è:

DISKCOMP [*d1:*] [*d2:*] [/1] [/8]

Il disco nel primo drive viene confrontato con quello nel secondo drive specificato: i due parametri possono indicare lo stesso drive (e in tal caso sarà necessario alternare i dischi quando richiesto) o essere omessi (viene utilizzato il drive di default).
Il comando DISKCOMP determina il numero delle facce e dei settori del disco nel primo drive, e si aspetta di trovare una configurazione simile nel secondo: potete però forzare il comando ad un confronto di un solo lato con l'opzione /1 o di otto settori per traccia con l'opzione /8.

Se il vostro disco è stato copiato per intero con il comando COPY *.*, di cui parleremo più avanti, può accadere che il confronto tra l'originale e la copia non dia esito positivo: il comando COPY, infatti, crea la disposizione più efficiente per i file copiati e ciò può provocare la non coincidenza tra le due strutture, anche se il contenuto dei file è rimasto invariato.

## PARTIZIONI DEL DISCO RIGIDO: IL COMANDO FDISK

Prima di poter utilizzare un disco rigido, è necessario riservare un'area, detta *partizione* del DOS che verrà poi formattata nel modo proprio al DOS e potrà contenere anche i file del sistema operativo (comando FORMAT).
Poiché la capacità di un disco rigido è notevolmente superiore a quella di un dischetto, in esso possono trovare spazio diversi sistemi operativi e poiché, come abbiamo già visto, ogni sistema utilizza formati diversi per i propri dati, il disco rigido deve essere suddiviso in sezioni separate le une dalle altre, ciascuna riservata per il formato di un dato sistema operativo.
Se intendete usare solo il sistema DOS, naturalmente la suddivisione ad esso dedicata può occupare l'intero disco; se invece vi occorrono diversi sistemi operativi, dovete innanzitutto stabilire quale parte del disco assegnerete a ciascuno di essi. Il comando FDISK vi permette di definire sul disco rigido solo la partizione del DOS: le altre saranno ottenute per mezzo di comandi simili appartenenti ai diversi sistemi operativi.
Molte sono le funzioni del comando FDISK: creare o cancellare la partizione sul disco rigido, cambiare la partizione corrente, esaminare lo stato del disco, predisporre un altro disco rigido su cui compiere operazioni.
Se nel vostro sistema è presente solo il DOS, dovete avviare il comando battendo FDISK, scegliere l'opzione CREATE DOS PARTITION (creazione della suddivisione del DOS) tra quelle presentate nel menu, rispondere Y (Sì) alla domanda DO YOU WISH TO USE THE ENTIRE FIXED DISK FOR DOS (intendete riservare tutto il disco rigido per il DOS): al termine di tutte queste operazioni siete pronti per formattare ed usare il vostro disco rigido.
Per avere una copia del sistema operativo DOS anche sul disco rigido specificate l'opzione /S nel comando FORMAT: sarete così in grado di caricare il DOS dal disco rigido. Il sistema, quando viene acceso o reinizializzato, dapprima controlla se nel drive A: ci sia un dischetto (e in tal caso questo deve contenere i file di sistema) da cui caricare il sistema operativo; nel caso in cui non lo trovi, il sistema si rivolge automaticamente al disco rigido e, se trova la partizione del DOS, carica da questa il sistema operativo.

## 3.6 Comandi DOS per il trattamento dei file su disco

Dopo aver predisposto i vostri dischi, vi servono dei comandi che vi permettano di trattare individualmente i file su disco: qui di seguito li esaminiamo uno per uno.

### IDENTIFICAZIONE DEL DISCO: IL COMANDO VOL

Il comando VOL (*Volume*) visualizza la volume label del disco presente nel drive richiesto, cioè il nome assegnato al disco per mezzo dell'opzione /V del comando FORMAT. La sintassi del comando è:

VOL [*d*:]

Come sempre, se non viene specificato il drive, viene visualizzata la label del drive di default. Per conoscere il nome assegnato al disco dovete quindi dare il comando:

```
A>VOL

 Volume in drive A is BASIC_PGMS
```

### CONTROLLO DEL CONTENUTO DI UN DISCO: IL COMANDO DIR

Il comando DIR, con la sintassi:

DIR [*specfile*] [/P] [/W]

visualizza l'elenco di tutti o alcuni file contenuti nel disco specificato e nel subdirectory indicato dal cammino fornito come argomento. Come al solito, se non viene indicato il drive, viene considerato quello di default e se non viene data la specificazione del subdirectory viene letto quello corrente.
Se nel comando DIR è presente un nome di file, vengono listati solo i file coincidenti con questo; possono essere utilizzati i caratteri jolly già visti per ottenere l'elenco di gruppi di file.
Se non viene indicato un nome, vengono elencati tutti i file presenti nel catalogo e nel drive adeguati, insieme alle dimensioni (in byte) ed alla data di creazione dei file. Vengono inoltre visualizzati la volume label e lo

spazio libero. Nell'esempio seguente richiediamo l'elenco di tutti i file del disco rigido appartenenti al subdirectory di nome BASIC e con estensione .BAS:

```
A>DIR C:\BASIC\*.BAS
 Volume in drive C is 800647
 Directory of  C:\BASIC

ART        BAS      1920    3-08-83   12:00p
SAMPLES    BAS      2304    3-08-83   12:00p
MORTGAGE   BAS      6272    3-08-83   12:00p
COLORBAR   BAS      1536    3-08-83   12:00p
MUSIC      BAS      8704    3-08-83   12:00p
DONKEY     BAS      3584    3-08-83   12:00p
CIRCLE     BAS      1664    3-08-83   12:00p
PIECHART   BAS      2304    3-08-83   12:00p
SPACE      BAS      1920    3-08-83   12:00p
BALL       BAS      2084    3-08-83   12:00p
COMM       BAS      4352    3-08-83   12:00p
       11 File(s)    8278016 bytes free

A>
```

Le opzioni /P e /W consentono un migliore controllo della visualizzazione: con /P la videata si ferma quando lo schermo è completo; per riprendere la visualizzazione, premete un qualsiasi tasto. L'opzione /W, invece, utilizza un formato più largo per i nomi, adatto ad un video con 80 colonne. Il comando DIR, come ricorderete, riconosce solo i file di dati e di programma presenti sul disco: i file nascosti di sistema, come IBMBIO.COM e IBMDOS.COM non vengono elencati.

## ESAME DEI CAMMINI: IL COMANDO TREE

Per esaminare la struttura ad albero di un dato disco, usate il comando TREE con la seguente sintassi:

TREE [*d:*] [/F]

Se lo specificatore di drive è omesso, viene utilizzato quello di default. Dopo aver esaminato il disco indicato, il comando TREE visualizza tutti i subdirectory trovati, con i relativi cammini. Specificando anche il parametro /F si ottiene anche l'elenco dei file in essi contenuti. L'esempio seguente mostra l'effetto del comando TREE sul catalogo proposto in Figura 3.1:

```
C>TREE A:

DIRECTORY PATH LISTING FOR VOLUME ???????????

Path: \VENDITE

Sub-directories:  MARIO
                  LUCIA

Path: \VENDITE\MARIO

Sub-directories:  None

Path: \VENDITE\LUCIA

Sub-directories:  None

Path: \CONTABIL

Sub-directories:  LUCA
                  GLORIA
                  CARLO

Path: \CONTABIL\LUCA

Sub-directories:  None

Path: \CONTABIL\GLORIA

Sub-directories:  None

Path: \CONTABIL\CARLO

Sub-directories:  None
```

## COME DEFINIRE CAMMINI ALTERNATIVI: IL COMANDO PATH

La maggior parte dei comandi DOS è in grado di esaminare solo il contenuto del drive e del catalogo specificati (o del drive di default e del subdirectory corrente) alla ricerca di altri comandi DOS o di blocchi di file. Se i file o il comando non vengono trovati dove indicato, la ricerca termina e viene visualizzato un messaggio d'errore.

Il comando PATH permette di indicare altri cataloghi a cui indirizzarsi se la ricerca nel catalogo specificato risulta infruttuosa. Eccone la sintassi:

PATH [*d:*] *cammino* [[;[*d:*] *cammino*]...]

I parametri possono essere uno o più cammini: un comando non trovato nel catalogo corrente sarà ricercato anche in tutti i subdirectory specificati dal comando ed eseguito non appena trovato. Se, per esempio, voleste raggruppare tutti i file con i comandi esterni del DOS in un subdirectory, chiamato DOS, del disco rigido corrispondente al drive C:, per eseguire i comandi stessi da un qualunque altro subdirectory date innanzitutto il seguente comando:

```
C>PATH C:\DOS
```

Per eliminare la lista dei cammini eseguite semplicemente il comando PATH; cioè il comando seguito dal solo punto e virgola. Se non specificate alcun parametro (e nemmeno il punto e virgola), verranno considerati i parametri specificati nel comando PATH precedente.

## CONTROLLO DELLO SPAZIO LIBERO IN MEMORIA E SU DISCO: IL COMANDO CHKDSK

Per mezzo del comando CHKDSK (*Check Disk*), con la sintassi:

CHKDSK [*specfile*] [/F] [/V]

potete ottenere una relazione dello stato della memoria e del disco indicato (o quello di default).
Dopo l'analisi del disco, il comando CHKDSK visualizza le seguenti informazioni:

— La volume label (il nome che avete assegnato al disco all'atto della formattazione) e la data di formattazione. Entrambe le informazioni vengono a mancare se non avete usato l'opzione /V nel comando FORMAT
— La quantità totale di memoria del disco
— Il numero di file nascosti (quelli presenti sul disco ma non elencati dal catalogo) e la parte di disco da essi occupata
— Il numero di subdirectory e la parte di disco da essi occupata
— Il numero di file dell'utente e la memoria che occupano
— Il numero di byte rimasti liberi

Oltre al disco, questo comando controlla sempre lo stato della memoria interna del PC e visualizza:

— Il numero totale di byte di memoria presenti
— Il numero di byte di memoria liberi, o comunque disponibili

Se tra i parametri del comando viene specificato anche il nome di un file, verrà aggiunto, alle altre informazioni, un resoconto sul numero di aree non contigue occupate da quel file sul disco: questo perché il DOS può essere stato costretto a suddividere il file in pezzi allocandoli in diverse parti del disco. Naturalmente il DOS ricostruisce automaticamente e in modo esatto un file di questo tipo, ma l'operazione di lettura richiede un tempo superiore al normale. (Fate riferimento al comando COPY in caso di difficoltà).
L'opzione /F vi permette di individuare e risolvere un particolare tipo d'errore sul disco: per svariate ragioni a volte un disco può contenere delle zone difettose. Queste aree sono identificate come inaccessibili per il DOS, mentre in realtà potrebbero essere usate normalmente; se il comando CHKDSK trova alcune di queste aree, vi chiede se volete recuperare i dati in esse contenuti, ponendoli in un file. Se avete il sospetto che almeno una parte del contenuto di queste zone di memoria sia importante, rispondete affermativamente alla domanda (Y, seguito dal tasto ENTER) e CHKDSK creerà uno o più file con estensione. CHK. Se invece vi interessa che questa area sia libera per altri scopi, rispondete N (No), e CHKDSK renderà di nuovo disponibili al DOS queste aree.
Tutti questi cambiamenti hanno luogo solamente in caso voi abbiate specificato l'opzione /F, altrimenti voi avrete una corretta analisi del problema ma non potrete fare alcuna correzione al disco.
Per poter seguire più in dettaglio le operazioni del comando CHKDSK, cioè sapere in ogni momento quello che sta facendo e che errori ha trovato, specificate il parametro /V. Confrontiamo ora l'effetto del comando CHKDSK con e senza l'opzione /V, per un dischetto simile a quelli degli esempi precedenti:

```
C>CHKDSK A:

    179712 bytes total disk space
     22016 bytes in 2 hidden files
      3584 bytes in 7 directories
     18944 bytes in 3 user files
    135168 bytes available on disk

    327680 bytes total memory
    279024 bytes free
```

```
C>CHKDSK A:/V
Directory A:
        A:\IBMBIO.COM
        A:\IBMDOS.COM
        A:\COMMAND.COM
Directory A:\VENDITE
Directory A:\VENDITE\MARIO
        A:\VENDITE\MARIO\APPUNTI.TXT
        A:\VENDITE\MARIO\ANALISI.TXT
Directory A:\VENDITE\LUCIA
Directory A:\CONTABIL
Directory A:\CONTABIL\LUCA
Directory A:\CONTABIL\GLORIA
Directory A:\CONTABIL\CARLO

    179712 bytes total disk space
     22016 bytes in 2 hidden files
      3584 bytes in 7 directories
     18944 bytes in 3 user files
    135168 bytes available on disk

    327680 bytes total memory
    279024 bytes free
```

## COME VISUALIZZARE I FILE DI TESTO: IL COMANDO TYPE

Per visualizzare il contenuto di un file residente su disco, potete usare il comando TYPE, la cui sintassi è:

TYPE *specfile*

Il DOS legge dal disco il file richiesto e ne visualizza il contenuto sullo schermo; per ottenere contemporaneamente una copia su carta, dovete controllare che la stampante sia pronta e premere i tasti CTRL e PRTSC appena prima di inviare il comando TYPE. Potete anche indirizzare l'output verso dispositivi diversi dal video e dalla stampante o anche a un qualsiasi file, per mezzo dei simboli > o >>, come segue:

```
C>TYPE MEMO.TXT>AUX:
```

Questo comando invia il contenuto del file MEMO.TXT all'interfaccia per comunicazioni asincrone (vedi anche "Redirezione dell'input e dell'output" più avanti in questo stesso capitolo).
Dal momento che il comando TYPE visualizza il file così come è scritto, non è molto conveniente utilizzarlo per file che non siano scritti in for-

mato ASCII, che risulterebbero in una forma assolutamente inintelleggibile: il comando TYPE BASICA.COM avrebbe un risultato di nessuna utilità, perché il file è scritto in linguaggio macchina e predisposto per essere letto ed eseguito direttamente dal microprocessore del PC. Solo i file precedentemente caricati (SAVE) con l'opzione ,A (cioè ASCII) vengono visualizzati in modo comprensibile dal comando TYPE.

## COME COPIARE UN FILE: IL COMANDO COPY

Il comando che permette di fare copie di uno o più file è COPY, con la sintassi standard:

COPY *specfile1* [*specfile2*] [/V]

Vedremo poco più avanti altre sintassi che consentono operazioni più complesse. I file presenti nello *specfile1* (detto file sorgente) vengono così copiati nel file indicato dallo *specfile2* (file destinazione).
Se nello *specfile2* non è contenuto un nome di file, il file sorgente viene copiato in un file con lo stesso nome nel drive specificato (o nel drive di default) e inserito nel catalogo indicato (o in quello corrente). La mancanza totale dello *specfile2* fa sì che il file venga copiato nel catalogo corrente del drive di default con lo stesso nome presente in *specfile1*.
Nel caso in cui il drive di default sia C: e il catalogo corrente \VENDITE, il comando:

```
C>COPY A:REPORT.APR
```

ha lo stesso effetto di questo, molto più complesso:

```
C>COPY A:REPORT.APR C:\VENDITE\REPORT.APR
```

cioè copia il file REPORT.APR dal drive A: al drive C: nel catalogo VENDITE.
L'opzione /V del comando COPY istruisce il DOS ad operare un controllo della copia: automaticamente rilegge la copia e la confronta con l'originale. Questa opzione rallenta l'esecuzione del comando COPY, ma è di indubbia utilità nella copia di file particolarmente critici.

NOTA: se avete in precedenza inviato il comando del sistema operativo VERIFY ON, questo controllo della copia verrà eseguito in ogni caso, anche senza opzione.

Potete sfruttare i caratteri jolly per copiare numerosi file di un unico

gruppo con un unico comando: se voleste copiare tutti i programmi in BASIC che risiedono nel catalogo corrente del drive di default (il drive C:), basterebbe il comando:

```
C>COPY *.BAS A:
```

che fa in modo che tutti i file nel catalogo corrente del drive C: che hanno come estensione .BAS siano copiati sul drive A:, mantenendo ciascuno il proprio nome.
Potete anche copiare tutti i file di un catalogo in un altro per mezzo del comando:

```
C>COPY *.* BACKUP
```

che copia tutti i file del catalogo corrente nel catalogo \BACKUP, sempre nel drive C:.
Il comando COPY cerca di riorganizzare i file nel modo più efficiente possibile, per sfruttare al massimo lo spazio del disco, cosa, questa, che il comando DISKCOPY non fa: se sospettate perciò che i vostri file stiano diventando troppo frammentati sul disco (controllate per mezzo del comando CHKDSK *.*), vi conviene copiarli, con il comando COPY *.*, su un nuovo disco appena formattato.

NOTA: se il disco contiene più di un directory, dovete copiare il contenuto di ciascuno esplicitamente.

### Concatenazione di file da copiare

Una variante spesso utile del comando COPY è quella che permette di combinare diversi file da copiare in uno solo: l'operazione, detta concatenazione, è eseguita in seguito ad un comando con questa sintassi:

COPY *specfile1* [+*specfile* [+*specfile*...]] [*specfile2*]

Tutti i file collegati dal segno + vengono letti uno di fila all'altro nell'ordine proposto e copiati nel file indicato da *specfile2*: i capitoli di un libro possono essere scritti separatamente e poi combinati con l'ausilio del comando:

```
C>COPY CAP1+CAP2+CAP3+CAP4+APPEND A:MIOLIBRO
```

Il file finale è MIOLIBRO, che si trova nel drive A: e contiene i capitoli dall'1 al 4 e l'appendice. In questo caso i file sorgente devono trovarsi nel

catalogo corrente del drive di default, poiché nella specificazione non viene indicato nessun cammino e nessun drive.
Mentre normalmente il comando COPY assegna al file copia la stessa data e ora di creazione del file sorgente, nel caso della concatenazione il file risultante porta la data e l'ora correnti.
Se nella specificazione del file destinazione tralasciate il nome del file, il comando COPY crea un file con il nome del primo dei file concatenati; nel caso in cui il primo file sorgente e il file destinazione si trovino nello stesso disco e catalogo, il primo file viene rimpiazzato da quello finale.
L'effetto del comando

```
C>COPY A:MIOLIBRO+C:APP2 A:MIOLIBRO
```

è perciò semplicemente quello di aggiungere una seconda appendice (file trovato nel drive C:) al file MIOLIBRO, appena creato.

### Indicazione del tipo di file da copiare: le opzioni /A e /B

Due opzioni molto utili, soprattutto nel caso della concatenazione sono le opzioni /A e /B, la cui sintassi è:

COPY [/A] [/B] *specfile1* [/A] [/B] [+*specfile* [/A] [/B]...]
*specfile2* [/A] [/B]

Quando viene assegnata ad un file sorgente, l'opzione /A (dove A sta per ASCII) fa in modo che questo venga copiato fino al primo carattere EOF (*End Of File*, cioè "fine del file"). Il carattere EOF è il carattere CTRL Z o il carattere esadecimale ASCII 1A ed indica che il file sorgente è terminato. Se invece l'opzione viene assegnata ad un file destinazione, viene aggiunto il carattere EOF al termine del file definitivo.
L'opzione /B, al contrario, è usata per copiare file binari (o non-ASCII): per quanto riguarda i file sorgente, essi vengono copiati interamente ed ogni eventuale carattere EOF viene considerato come un carattere qualsiasi; nei file destinazione non viene aggiunto il carattere EOF al termine della copia.
Le opzioni /A e /B seguono la specificazione del file a cui si riferiscono direttamente, e restano esecutive fino a quando non viene indicata una nuova opzione. In realtà per default viene utilizzata l'opzione /A nel caso della concatenazione e l'opzione /B nel caso di copia semplice.
Nel caso della concatenazione, il file sorgente viene normalmente copiato fino al carattere EOF e ciò non causa alcun problema nel caso di file di tipo testo (scritti in formato ASCII), ma potrebbe procurarne per file binari, come quelli che contengono il programma oggetto.

Un file non-ASCII alla cui specificazione sia aggiunta l'opzione /B verrà copiato interamente; se l'opzione segue un file destinazione non verrà aggiunto il carattere EOF al termine della copia.

### Il comando COPY e i dispositivi del PC

Potete sfruttare il comando COPY anche per trasferire file attraverso tutti i dispositivi ausiliari del PC elencati nella Tabella 3.1. Se, per esempio, voleste creare direttamente dalla tastiera un file residente su disco, dovreste comandare al DOS di copiare direttamente dalla tastiera (dispositivo il cui nome simbolico in DOS è CON:) al file destinazione, per mezzo del comando:

```
C>COPY CON: DIRECT.TXT
```

e quindi battere, linea dopo linea, il vostro file che verrà contemporaneamente copiato nel file destinazione DIRECT.TXT. Per terminare la copia è sufficiente premere il tasto funzione F6 o i tasti CTRL Z, corrispondenti al carattere EOF che va a concludere il file DIRECT.TXT.
I dati battuti alla tastiera possono essere anche inviati ad una delle porte di comunicazione (ad esempio, COM1:) con il comando:

```
C>COPY CON: COM1:
```

Per concludere, questo comando vi permette anche di stampare un file inviandolo al dispositivo LPT1:

```
C>COPY MIOLIBRO.TXT LPT1:
```

è il comando che dà inizio alla stampa del file creato precedentemente.

## VERIFICA DELLA COPIA: IL COMANDO VERIFY

La maggior parte dei dati che scrivete su un disco saranno memorizzati esattamente come vi aspettate, grazie all'alto grado di affidabilità dei supporti magnetici; purtroppo, a volte si verificano degli imprevisti e dati che voi pensate al sicuro non risultano registrati correttamente.
Abbiamo appena esaminato l'opzione /V del comando COPY che permette di verificare la copia di un file, ma per controllare tutte le operazioni di scrittura (e non solo la copia) è necessario un nuovo comando:

VERIFY [ON|OFF]

La richiesta di VERIFY ON istruisce il DOS a rileggere tutti i dati appena dopo la scrittura ed a confrontarli con l'originale per verificare che l'operazione sia avvenuta senza errori: questo naturalmente rallenta tutte le operazioni di scrittura su disco.
Se vi accorgete che i vostri dati vengono persi con una certa regolarità usate il comando VERIFY solo come soluzione temporanea: considerate invece l'opportunità di usare dischi di qualità migliore o di far controllare il vostro drive.
Il comando VERIFY OFF termina il processo di verifica, mentre il solo VERIFY (senza ON né OFF) ha come unico effetto la visualizzazione dello stato corrente del comando.

## CONFRONTO TRA FILE: IL COMANDO COMP

Il comando COMP vi permette di confrontare due file, per assicurarvi che essi siano identici; la sintassi è la seguente:

COMP [*specfile1*] [*specfile2*]

Se tralasciate entrambe le specificazioni, queste vi verranno richieste successivamente dal comando COMP: è questa una procedura utile nel caso in cui vogliate confrontare due file che si trovino su dischetti diversi, nessuno dei quali sia un disco di sistema. In questo caso, inserite il disco DOS nel drive A:, battete A:COMP, seguito dal tasto ENTER; alla richiesta dei nomi dei file, estraete il disco dal drive A: ed inserite i due dischetti nei drive A: e B:; infine battete le specificazioni dei due file per completare il comando.
Se viene invece indicata solo la prima specificazione, il comando assume che il secondo file abbia lo stesso nome e la stessa estensione del primo, e si trovi nel catalogo corrente del drive di default.
Nel nome del file possono essere utilizzati i caratteri jolly * e ?; tutti i file in accordo con il primo nome verranno confrontati con i corrispondenti. Il comando:

```
C>COMP A:CAPIT?.* C:*.*
```

confronterà tutti i file del drive A: (ma solo nel catalogo corrente) il cui nome inizia con CAPIT (potrebbero essere CAPIT1.TXT, CAPIT2.TXT, CAPIT2.BAK) con i file identificati dallo stesso nome e dalla stessa estensione che si trovano nel catalogo corrente del drive C:.
Il confronto viene eseguito byte per byte e se una coppia di byte non risulta identica viene visualizzata la posizione dell'errore (cioè la locazione

del file in cui si trova il byte) e il valore dei due byte che non coincidono. Dopo dieci di questi errori l'azione del comando termina.

## COME CAMBIARE IL NOME DI UN FILE SU DISCO: I COMANDI RENAME E REN

I due comandi RENAME e REN hanno l'identico effetto di cambiare il nome di un file già esistente. La loro sintassi è la seguente:

RENAME *specfile nomefile* [*.est*]

oppure

REN *specfile nomefile* [*.est*]

ed il risultato è di cambiare il nome contenuto in *specfile* con *nomefile*. Non potete usare il comando per trasferire file da un disco ad un altro o da un catalogo ad un altro: l'unico cambiamento riguarda il nome contenuto nel catalogo del disco.

## COME CANCELLARE I FILE: I COMANDI ERASE E DEL

Altri due comandi con effetto identico sono ERASE e DEL che consentono di cancellare file presenti sul disco. La sintassi è:

ERASE [*specfile*]

oppure

DEL [*specfile*]

ed il risultato è la eliminazione dal disco del file indicato. Anche in questo caso potete usare i caratteri jolly sia per il nome che per l'estensione del file. Il comando

```
C>DEL *.*
```

ha il drastico effetto di cancellare tutti i file che si trovano nel catalogo corrente del drive di default; se usate questo parametro, prima di eseguire il comando il DOS vi chiederà se siete sicuri di voler intraprendere questo passo.

NOTA: entrambi i comandi DEL ed ERASE non eliminano un catalogo. Per cancellare un catalogo vuoto dovete usare il comando RMDIR.

## SALVATAGGIO DEI DATI DI UN DISCO RIGIDO: I COMANDI BACKUP E RESTORE

Mentre un floppy disk ha una capacità variabile tra 180000 e 360000 byte, un disco rigido può avere una capacità di parecchi megabyte: in questo modo, per avere una copia di sicurezza dell'intero contenuto del vostro disco rigido, vi occorrono decine di dischetti e tanto tempo da dedicare all'operazione!
Il DOS, però, vi fornisce due comandi, BACKUP e RESTORE, che vi facilitano questo compito gravoso. Al comando BACKUP corrisponde un meccanismo flessibile che vi consente di selezionare i file dal disco rigido e di inviarli ad uno o più dischetti. La sintassi per questo comando è:

BACKUP [*specfile*] *d:* [/S] [/M] [/A] [/D:*mm-gg-aa*]

Tutti i file la cui specificazione corrisponde a quella inserita nel comando saranno copiati sul dischetto presente nel drive indicato (*d:*); i dischetti devono naturalmente essere stati precedentemente formattati.
Come per la maggior parte dei comandi DOS, anche in questo caso verranno usati il catalogo corrente ed il drive di default se non indicato altrimenti; possono inoltre essere utilizzati i caratteri jolly * e ? nel nome del file. Tralasciare il nome del file equivale ad usare la notazione *.*.
I parametri opzionali consentono una grande versatilità nell'effettuazione del backup: il parametro /S cerca i file non solo nel catalogo indicato ma anche in ogni suo subdirectory, scendendo attraverso tutti i livelli. Con questa facilitazione, il comando per copiare l'intero disco rigido del drive C: diviene il seguente:

```
C>BACKUP C:\*.* A:/S
```

La copia parte cioè dalla radice (indicata dal simbolo \) e interessa tutti i file (indicati dal simbolo *.*) dipendenti dalla radice e da tutti i subdirectory (come indicato dall'opzione /S). In questo esempio, anzi, la notazione *.* sarebbe facoltativa: il comando avrebbe esattamente lo stesso effetto se nessun nome venisse indicato.
Il parametro /M è molto utile nel caso voi intendiate salvare periodicamente il contenuto del disco rigido: con questa opzione, infatti vengono copiati solo i file che sono stati modificati dopo l'esecuzione del precedente comando BACKUP e così i file, come BASIC.COM, che non vengono mai, o quasi mai modificati, possono essere copiati una volta per sempre.

Nella versione normale, il comando BACKUP cancella il contenuto del dischetto prima di scriverci sopra: l'opzione /A permette di evitare questa operazione, in modo da poter aggiungere file ad altri già esistenti e usare così lo stesso dischetto per numerose esecuzioni del comando BACKUP. Se vi interessa salvare solo i file creati o modificati dopo una certa data, vi è utile l'opzione /D, con la sintassi D: mese-giorno-anno; così, se volete copiare tutti i file del \ VENDITE \ MAGGIO creati o modificati dall'inizio del mese di maggio, usate il comando:

```
C>BACKUP C:\VENDITE\MAGGIO A:/S/D:05-01-83
```

Come potete notare, le due opzioni /S e /D vi consentono di copiare anche i file dei subdirectory di \ VENDITE \ MAGGIO modificati o creati dopo il primo giorno di maggio.
Durante la copia di ogni file ne viene visualizzato il nome; quando un dischetto è pieno, BACKUP vi richiede di inserirne uno nuovo: ricordatevi di prendere nota dell'ordine in cui sono stati scritti i dischetti, poiché questo è di vitale importanza nel recuperare i dati successivamente.
Per questa successiva operazione si utilizza il comando inverso di BACKUP, cioè RESTORE, che consente di recuperare un file da un disco creato per mezzo del comando BACKUP e di scriverlo su un disco rigido. La sintassi è:

RESTORE *d:* [*specfile*] [/S] [/P]

in cui l'argomento *d:* indica il drive da cui vanno letti i file e *specfile* indica il disco rigido, il catalogo e i file che devono essere recuperati. L'omissione di *specfile* corrisponde all'uso del drive di default, del catalogo corrente e della notazione *.*.
Come nel comando precedente, anche qui possono essere usati i caratteri jolly * e ? nel nome e nell'estensione dei file; per esempio

```
C>RESTORE A: C:\TEXT\*.TXT
```

è il comando che richiede il recupero di tutti i file con estensione .TXT che originariamente si trovavano nel catalogo \ TEXT.
Il comando RESTORE può copiare da un dischetto in un determinato catalogo del disco rigido solo quei file che in precedenza erano stati letti da quel catalogo e scritti su disco per mezzo del comando BACKUP: se dopo aver copiato l'intero contenuto di un catalogo, cambiate il catalogo corrente ed eseguite il comando RESTORE senza specificare il catalogo, nessun file verrà recuperato, poiché nessuno di quelli salvati proviene dal catalogo corrente.
L'opzione /S vi consente di recuperare anche tutti i file contenuti nei sub-

directory del catalogo indicato; se uno dei subdirectory è stato cancellato dopo il BACKUP (comando RMDIR) per mezzo dell'opzione /S, il comando RESTORE lo ricrea e vi copia tutti i file originariamente presenti.
L'opzione /P comanda al DOS di richiedervi il nome di ognuno dei file di backup e vi permette quindi di selezionare quello su cui compiere l'operazione. Questo parametro viene usato nel caso di file che siano stati corretti dopo il backup o indicati come file a sola lettura.
Il comando RESTORE si attende i dischetti nello stesso ordine in cui essi sono stati scritti dal comando BACKUP; quando un dischetto viene copiato, vi viene richiesto di inserire il successivo, se necessario.

## RECUPERO DI FILE DA UN DISCO DIFETTOSO: IL COMANDO RECOVER

In qualche occasione può accadere che su uno dei dischi che state utilizzando, un settore abbia un comportamento difettoso; in una tale situazione vi sarà d'aiuto il comando RECOVER, la cui sintassi è:

RECOVER *specfile*

oppure

RECOVER *d:*

Le due diverse forme del comando RECOVER producono due diversi effetti: la prima viene utilizzata quando il settore difettoso fa parte di uno dei file memorizzati. In questo caso il comando RECOVER ricostruisce l'intero file all'infuori del settore in questione.
Poiché un settore contiene 512 byte, ogni volta che un file in cui è presente un settore difettoso viene recuperato perde una parte di testo lunga 512 caratteri; inoltre a volte vengono aggiunti caratteri casuali alla fine del file recuperato. Se il file è un file di testo potete eliminare subito questi eventuali intrusi con l'ausilio del comando EDLIN.
La seconda forma invece è necessaria in tutti i casi in cui il settore difettoso faccia parte del catalogo del disco: specificate allora il drive contenente il disco e verranno recuperati tutti i file in esso presenti.
Tutto il catalogo è inaffidabile anche se un solo settore di esso non è corretto: per questo motivo non vengono più usati i nomi originali dei file, ma nomi standard la cui forma è FILE*nnnn*. REC, dove *nnnn* è un numero progressivo, a partire da 0001. Tutti i nomi e le estensioni originali vengono persi e a voi rimane lo sgradito compito di stabilire il contenuto di ogni singolo file: vi consigliamo perciò di usare il comando RECOVER solo quando è strettamente necessario.

Dopo l'esecuzione del comando, sia su un file che su catalogo, dovete copiare tutti i file di quel disco che vi interessano su un altro disco e riformattare il disco che presentava il settore difettoso. (Il comando FORMAT, come abbiamo visto, rintraccia i settori difettosi e li segnala in modo da renderli inaccessibili al DOS per tutti gli usi successivi).

## 3.7 Altri comandi DOS

Oltre ai comandi che consentono la creazione e la manipolazione dei file, il DOS possiede numerosi comandi che assolvono un gran numero di compiti.

### COME SELEZIONARE I MODI DI SISTEMA: IL COMANDO MODE

Il PC è un sistema molto flessibile: per sfruttarlo in pieno l'utente deve essere in grado di cambiare o riconfigurare i diversi dispositivi che completano il sistema. Il comando MODE fornisce un meccanismo standard per selezionare e cambiare il modo per le operazioni con stampante, video e interfaccia asincrona. Sono quattro le configurazioni possibili del comando MODE.

#### Modo di stampa

Questa versione del comando predispone il modo di operazione con le stampanti, da LPT1: a LPT3:

MODE LPT#:[*n*][, [*m*][,P]]

Un esempio:

```
C>MODE LPT1:132,8
```

Il parametro *n* seleziona il numero di caratteri per linea (80 oppure 132), mentre *m* stabilisce la spaziatura verticale, misurata in linee per pollice (6 oppure 8). Il parametro P, se specificato, indica di riprovare continuamente, anche in caso di errore di *timeout* (questo tipo di errore si verifica quando un dispositivo, come una stampante, non risponde al PC entro un definitivo intervallo di tempo dopo la richiesta del PC stesso). Nel modo specificato dal parametro P, il PC continuerà a richiedere la risposta della stampante anche se questa non fosse, ad esempio, collegata.

I valori di default della stampante sono 80 caratteri per riga e 6 linee per pollice; nell'esempio precedente la stampante n.1 viene predisposta a stampare 132 caratteri per riga e 8 linee per pollice. Nel momento in cui inviate questo comando la stampante dev'essere accesa, altrimenti le selezioni non avranno effetto ed il PC segnalerà l'errore; nello stesso modo, quando spegnete la stampante essa riassumerà i valori di default.

### Abilitazione del video

Questa versione permette di selezionare l'adattatore video richiesto o il modo video per la scheda Colore/Grafica:

MODE *n*

oppure

MODE [*n*],*m*[,T]

Un esempio:

```
C>MODE CO80,R
```

Il parametro *n* abilita un particolare monitor e predispone uno specifico modo. Le scelte possibili sono le seguenti:

- Per la scheda Colore/Grafica: 40, 80, BW40, BW80, CO40 oppure CO80, dove 40 e 80 indicano il numero di caratteri per riga, BW indica bianco e nero, CO indica colore.
- Per l'adattatore monocromatico: MONO abilita il video monocromatico, che è sempre predisposto per 80 caratteri per riga.

Il parametro *m* può assumere i valori R o L, ed è usato per far scorrere la videata a colori a destra (R) o a sinistra (L) di un carattere, nel caso che una parte di ciò che volete visualizzare risulti esclusa dallo schermo. (Potete far eseguire questo MODE tante volte quante vi servono per spostare anche di diversi caratteri la videata). Il parametro T, infine, permette di controllare l'allineamento della videata e di spostarla ancora se occorre.
Nell'esempio abbiamo abilitato il video nel modo a colori con 80 caratteri per riga, ed abbiamo spostato la videata a destra di un carattere. Vi sottolineiamo che il modo colore non fa apparire il testo a colori, ma semplicemente prepara l'hardware ad utilizzare la possibilità del colore se il software ne richiede l'uso.

## Predisposizione dell'adattatore per monitor monocromatici

Questa versione fa in modo che l'output sia inviato all'adattatore per monitor monocromatici:

MODE COM*n:baud*[,*parità* [,*bit dati*[,*bit stop*[,P]]]]

Un esempio:

```
C>MODE COM1:30,N,8,,P
```

Il parametro *n* specifica il numero dell'adattatore (1 o 2), *baud* può assumere i valori 110, 150, 300, 600, 1200, 2400, 4800 o 9600 ed indica il *baud rate* (la velocità di trasmissione) che l'adattatore deve usare. (Sono sufficienti le prime due cifre del numero, così 96 sta per 9600 e 30 sta per 300).
I parametri *parità* (N=nessuna, O=dispari, E=pari, con E valore di default), *bit dati* (7 o 8, con 7 valore di default) e *bit stop* (1 o 2, con 1 solitamente valore di default) sono parametri standard richiesti per compiere operazioni con porte seriali; per maggiori dettagli su questi parametri, vedere il Capitolo 11. Anche in questo caso l'opzione P indica il continuo tentativo di superare gli errori di tipo timeout. Il nostro esempio seleziona il modo dell'adattatore COM1 a 300 baud, nessuna parità, otto bit di dati ed 1 bit di stop (come da default). Gli errori di timeout vengono continuamente testati.

## Redirezione dell'output di stampa

Questa versione del comando permette di indirizzare alla scheda di interfaccia *n* (*n*=1 o 2) l'output che era stato precedentemente inviato alla stampante parallela # (# =1,2 o 3).
Il comando appare:

MODE LPT#:=COM*n*

In questo modo, se avete collegato una stampante seriale all'adattatore COM1, l'esempio che segue invierà alla stampante seriale tutti gli output che vengono indirizzati al dispositivo LPT1:.

```
C>MODE LPT1:=COM1
```

NOTA: prima di redirigere l'output assicuratevi di aver inizializzato la scheda di interfaccia per mezzo della terza variante del comando MODE. Notate anche che la prima versione, il modo di stampa, elimina tutte le istruzioni di redirezione: siate certi, perciò, di aver ribattuto tutti i comandi che vi servono.

## COME STAMPARE CON IL DOS: I COMANDI GRAPHICS E PRINT

Se possedete una stampante grafica IBM (o una stampante Epson MX) potete usare il comando GRAPHICS che invia una copia (*dump*) della grafica presente sullo schermo, dalla scheda Colore/Grafica, alla stampante suddetta. (Per il testo normale, questo comando è decisamente superfluo).
Dopo aver inizializzato il sistema eseguite una sola volta il comando

```
GRAPHICS
```

che carica nella memoria RAM del vostro PC uno speciale programma che rimarrà residente finché non spegnete il computer o reinizializzate il sistema.
Ogni volta che premete i tasti SHIFT PRTSC per far stampare la videata, il programma residente controlla se sia necessario operare una copia della grafica: nel caso sia presente solo del testo, questo verrà stampato normalmente, come abbiamo visto in precedenza, mentre la grafica sarà adeguatamente riprodotta, in un tempo che può durare fino a tre minuti. (Il paragrafo 9.14, "Come stampare una videata grafica" vi fornisce una guida per utilizzare questo comando dall'interno di un programma BASIC). Per stampare, invece, file di tipo testo è disponibile il comando PRINT, che vi permette di accodare fino a 10 file in attesa di essere stampati, mentre il PC continua ad eseguire le normali operazioni del DOS (anche se in questo modo la stampa rallenta considerevolmente). Fate riferimento all'Appendice D per maggiori dettagli sul comando PRINT.

## COME CANCELLARE LO SCHERMO: IL COMANDO CLS

Per cancellare completamente dallo schermo qualsiasi testo vi sia riportato, battete il comando DOS:

```
CLS
```

### COME CONTROLLARE LA VERSIONE DEL DOS: IL COMANDO VER

Se non siete sicuri di quale versione del DOS sia in esecuzione, potete inviare il comando

```
VER
```

La risposta, nel caso della versione 2.0 sarà:

```
IBM Personal Computer DOS Version 2.00
```

### COME USCIRE DALL'ESECUZIONE DI UN PROGRAMMA: IL COMANDO BREAK

Il DOS consente l'utilizzo di due modi di risposta alla richiesta di interruzione di un programma; normalmente, quando un programma è in esecuzione, il DOS controlla se avete premuto i tasti CTRL BREAK solamente durante le operazioni di accesso allo schermo, alla tastiera, alla stampante o a dispositivi ausiliari.
Dal momento che la maggior parte dei programmi esegue almeno una di queste operazioni con una certa frequenza, il comando di CTRL BREAK verrà eseguito in un tempo ragionevolmente breve. In alcuni casi, però, il programma deve eseguire molte operazioni di calcolo e può passare molto tempo prima che venga eseguita una delle operazioni suddette. Se usate uno di questi programmi potete inviare il comando

BREAK [ON|OFF]

in cui BREAK ON fa in modo che il DOS controlli un eventuale CTRL BREAK ad ogni chiamata di una funzione DOS, aumentando così le probabilità che venga eseguito in un tempo breve, nonostante l'esecuzione di tutti i programmi risulti leggermente rallentata.
BREAK OFF riporta il sistema nella condizione precedente (di default), mentre BREAK, senza alcun parametro, visualizza la situazione corrente del modo BREAK.

## 3.8 Comandi avanzati per la gestione del DOS

Oltre ai comandi DOS più comuni esaminati finora esistono altri comandi che permettono di aumentare la potenza del sistema operativo DOS; non ne faremo una trattazione completa, ma più semplicemente faremo

una veloce panoramica poiché in realtà pochi utenti hanno necessità di usarli.

## REDIREZIONE DELL'INPUT E DELL'OUTPUT

L'espressione "input ed output" o semplicemente I/O si riferisce ai metodi utilizzati dal PC per la comunicazione dei dati in ingresso ed in uscita. La tastiera è il dispositivo standard di input (il mezzo attraverso cui voi inserite dati nel PC) ed il video è il dispositivo standard di output (attraverso cui ricevete la maggior parte delle informazioni che vi vengono dal PC).
Normalmente un programma è fatto per ricevere input dalla tastiera, ma potreste volergli far leggere i dati da disco; oppure vi potrebbe servire inviare i dati di output a un dispositivo diverso dal video, per esempio ad un modem per inviarli ad un altro computer. La capacità di variare il dispositivo di input e output è detta *redirezione*.
Il DOS vi consente la redirezione dell'I/O nella maggior parte dei programmi; una delle seguenti specificazioni dev'essere a tal fine aggiunta alla linea che contiene il comando DOS:

>[*d:*] [*cammino*] *nomefile*

invia l'output al dispositivo o al file specificato. Se il nome non coincide con quello di un dispositivo valido o di un file esistente, viene creato un file con quel nome, altrimenti il contenuto del file esistente viene cancellato prima che l'output sia scritto in quel file. (Vedi Tabella 3.1 per i nomi dei dispositivi).
Il comando

>>[*d:*] [*cammino*] *nomefile*

ha lo stesso effetto del precedente, tranne per il fatto che, se il nome corrisponde a quello di un file già esistente, questo non viene cancellato e al suo contenuto viene aggiunto l'output del programma.
Inoltre

<[*d:*] [*cammino*] *nomefile*

fa sì che l'input venga ricercato dal dispositivo o dal file specificato, piuttosto che dalla tastiera.
Vi ricordiamo che questo tipo di redirezione funziona solo con i programmi che utilizzano il DOS per effettuare l'input e l'output dei dati.
Come esempio, vi proponiamo il comando che permette di stampare l'intera struttura del catalogo e tutti i file contenuti nei subdirectory di un

disco: il comando TREE con le opzioni /F (file) e PRN ad indicare il nome del dispositivo di stampa (il segno di due punti è facoltativo):

```
TREE /F > PRN
```

## COLLEGAMENTO "PIPELINE" DI I/O

Un *pipe* è uno strumento software che permette di collegare assieme due o più programmi: potete cioè fare in modo che l'output di un programma venga usato come input di un secondo programma. In questo caso risulta chiaro il senso in cui viene usato il termine inglese *pipe*, che significa conduttura: l'output del primo programma viene forzato come attraverso una conduttura e diretto a divenire l'input di un altro programma.
Supponete di avere il programma DATA_MAN che produca una quantità considerevole di output numerico, ma che a voi, per una qualche ragione, interessino solo quei valori che superano 54.332. Potete scrivere un semplice programma (chiamiamolo DATA_AID) che legge dal dispositivo di input (usualmente la tastiera), controlla ogni valore in input per stabilire se sia maggiore di 54.332 e infine visualizza solo i valori che posseggono il requisito richiesto.
Dopo aver scritto il programma DATA_AID, potete collegare in pipeline l'output di DATA_MAN in modo che venga usato come input per DATA_AID: vedrete così sullo schermo solo i valori maggiori di 54.332. Per ottenere questo collegamento battete i nomi dei due programmi, separati dal simbolo (¦); nelle specificazioni dei file vanno aggiunti tutti i parametri necessari. Possono essere collegati in questo modo più file, utilizzando una sola linea per il comando. Nel nostro caso scriviamo:

```
DATA_MAN ¦ DATA_AID
```

## FILTRI DOS: SORT, FIND E MORE

Il programma DATA_AID usato prima è un tipico esempio di programma "filtro": un programma filtro legge i dati dal dispositivo standard di input, esegue alcune manipolazioni su questi dati ed infine scrive il risultato sul dispositivo di output. DATA_AID è un filtro che legge i valori dal dispositivo di input e visualizza solo quelli maggiori di 54.332.
Questi programmi non possiedono alcunché di intrinsecamente speciale, ma sono semplicemente dei programmi di impiego generale per l'utente DOS. Un filtro risulterebbe essere un semplice programma che ha l'unica funzione di lavorare con dati provenienti dal dispositivo standard di input e da inviare al dispositivo standard di output, ma la capacità del

DOS di redirigere l'I/O standard fa del filtro uno strumento ben più potente.
Tre filtri con caratteristiche generali sono presenti nel DOS: SORT, FIND e MORE, le cui sintassi sono:

SORT [/R] [/+*n*]

SORT è il programma che legge una linea dopo l'altra dal dispositivo di input, le mette in ordine alfabetico e invia le linee ordinate al dispositivo di output. L'opzione /R predispone un ordinamento inverso (dalla Z alla A) e l'opzione /+*n* permette di scegliere la colonna in cui ha inizio l'operazione di ordinamento.

FIND [/V] [/C] [/N] *"stringa"* [*specfile...*]

FIND ricerca tra i dati che provengono dall'input le linee che contengono la *stringa* indicata. (Una stringa è una sequenza di caratteri racchiusa tra virgolette; verrà discussa ampiamente nel Capitolo 5). Ogni linea che contiene la stringa cercata viene inviata all'output. Con l'opzione /V vengono inviate le linee che non contengono la stringa; l'opzione /C genera il numero di corrispondenze trovate, invece che il testo delle linee; il parametro /N, infine, visualizza il numero della linea davanti alla linea stessa.

MORE

MORE ha il semplicissimo compito di leggere linee dal dispositivo di input ed inviarle direttamente al dispositivo di output. Dopo aver riempito una "pagina", il programma viene sospeso, visualizza il messaggio —MORE— ed attende che l'utente prema un qualsiasi tasto. Quando viene premuto un tasto l'operazione riprende fino ad esaurimento dell'input.

NOTA: tutti e tre questi filtri possono sfruttare la redirezione dell'I/O standard, in modo da poter ricevere input ed inviare output ad ogni dispositivo o file.

Per dimostrare quanto siano utili questi filtri, supponiamo di voler leggere un catalogo, ordinarlo a seconda della lunghezza dei file ed inviare l'elenco ad un file di nome FILESIZE: tutto ciò può essere fatto con il semplice comando

```
DIR | SORT /+14 > FILESIZE
```

### CONTROLLO ESTERNO DEL SISTEMA: IL COMANDO CTTY

Per mezzo del comando CTTY potete escludere la tastiera ed il video standard del PC ed usare un terminale esterno per il controllo dell'input e dell'output. La sintassi del comando è la seguente:

CTTY *dispositivo*

dove il *dispositivo* può essere AUX, COM1, COM2 o un altro qualsiasi dispositivo *character-oriented.* Per ripristinare la tastiera e il video come standard I/O, inviate di nuovo il comando CTTY, ma senza alcun parametro per il dispositivo esterno. Naturalmente il dispositivo scelto dev'essere in grado di compiere operazioni sia di input che di output; per questo, una stampante non sarebbe adatta, essendo esclusivamente un dispositivo di output.

### COME PERSONALIZZARE IL PROMPT: IL COMANDO PROMPT

Potete cambiare il prompt standard del DOS (A>, B>, C> e così via) con una grande varietà di simboli: potete, ad esempio, includere nel prompt l'indicazione del catalogo corrente del drive di default con il comando

```
PROMPT $p$g
```

Per ulteriori dettagli, fate riferimento al paragrafo relativo al comando PROMPT nell'Appendice D.

## 3.9 File batch

Il DOS permette di raggruppare una serie di comandi in un unico file, chiamato *file batch,* che può essere mandato in esecuzione battendone il nome come se fosse un comando esterno del DOS: il sistema allora esegue uno dopo l'altro tutti i comandi che trova nel file.
I file batch sono caratterizzati dall'estensione che deve sempre essere .BAT; nel battere il nome del file per l'esecuzione, comunque, come per i comandi esterni del DOS, va tralasciata l'estensione.
Ogni comando del file viene eseguito come se l'aveste battuto da tastiera e potete interromperne l'esecuzione semplicemente premendo i tasti CTRL BREAK: il DOS vi chiederà conferma delle vostre intenzioni

```
Terminate batch job (Y/N)?
```

Se battete Y, il resto del file non viene eseguito e riappare il prompt caratteristico del DOS, se invece rispondete N viene fermato solo il comando in esecuzione ed il sistema passa al successivo nel file.

## COME CREARE UN FILE BATCH

Vediamo ora come creare un file batch: dapprima decidiamo quali compiti debba eseguire il DOS. Supponiamo che siano i seguenti:

1. Inserire la nuova ora
2. Inserire la nuova data
3. Avvisare l'operatore di inserire un disco nel drive B:
4. Visualizzare il directory del disco nel drive B:
5. Visualizzare un messaggio per l'operatore
6. Copiare un programma dal drive B: al drive A:
7. Avviare l'Advanced BASIC ed eseguire il programma appena copiato

Per l'intera operazione è necessaria la sequenza di comandi qui riportata:

```
TIME
DATE
PAUSE Inserire nel drive B: il disco da cui copiare
DIR B:
REM Ora viene effettuata la copia
COPY B:TOPOLINO.BAS A:PAPERINO.BAS
BASICA A:PAPERINO.BAS
```

Prima di creare il file, esaminiamo in breve i quattro nuovi comandi DOS che stiamo per usare

### I comandi TIME e DATE

I comandi TIME e DATE, che potete inviare sia da tastiera che dall'interno di un file batch, vi permettono di regolare ora e data del sistema. La sintassi è la seguente:

TIME
DATE

Entrambi questi comandi vi richiedono esplicitamente le informazioni necessarie.

### Il comando PAUSE

Il comando PAUSE consente di sospendere temporaneamente l'esecuzione di un file batch e nello stesso tempo visualizza un messaggio; può essere usato solo dall'interno di un file batch, con la sintassi:

PAUSE [*messaggio*]

L'esecuzione di questo comando provoca la visualizzazione del *messaggio* inserito (che potrebbe essere "Inserire nel drive B: il disco da cui copiare") oltre al messaggio standard STRIKE ANY KEY WHEN READY (Premi un tasto qualsiasi quando sei pronto). Il DOS resta in attesa finché non premete un tasto per avvisarlo di aver compiuto l'azione richiesta, e quindi prosegue con il comando successivo.
Il comando PAUSE facilita l'interruzione di un file batch ad un punto predeterminato: quando viene visualizzato il messaggio PAUSE, basta premere CTRL BREAK per ritornare al DOS.

### Il comando REM

Il comando REM, molto simile al precedente, tranne per il fatto che non interrompe l'esecuzione del file batch, permette di visualizzare il messaggio scelto, da inserire in questo modo:

REM [*messaggio*]

La frase può essere composta da 123 caratteri al massimo; il comando REM può essere utilizzato anche senza messaggio solo per aumentare la comprensibilità del file.

### Esecuzione del file batch

Potete creare il file usando il comando COPY per trasferire il testo direttamente dalla tastiera ad un file su disco. Se perciò volete creare il file batch ROBOT.BAT contenente la lista di comandi sopra elencata, innanzitutto battete:

```
COPY CON: ROBOT.BAT
```

e poi i comandi, esattamente come segue:

```
TIME
DATE
PAUSE Inserire nel drive B: il disco da cui copiare
DIR B:
REM Ora viene effettuata la copia
COPY B:TOPOLINO.BAS A:PAPERINO.BAS
BASICA A:PAPERÌNO.BAS
```

Dopo aver battuto l'ultimo comando che deve andare a far parte del file batch, premete il tasto funzione F6 e ENTER per terminare e far scrivere su disco il file appena inserito.
Ora, per mandare in esecuzione il file, basta semplicemente batterne il nome, senza estensione:

```
ROBOT
```

Il DOS vi risponderà subito, incominciando ad eseguire i comandi uno alla volta.

## FILE BATCH CON PARAMETRI SOSTITUIBILI

Potete creare un file batch contenente parametri fittizi che verranno sostituiti, al momento dell'esecuzione, dai valori che voi avrete inserito a seguito del nome del file batch.
Un file può contenere fino a dieci di questi parametri fittizi, identificati da label che vanno da %0 a %9: il parametro %0 è particolare, in quanto assume sempre, durante l'esecuzione, il nome del file stesso.
Ecco un esempio di come creare un file batch con parametri fittizi:

```
COPY CON: ARTIFINT.BAT
TYPE %0
COPY %1.TXT %2.TXT
COMP %1.TXT %2.TXT
BASICA %3.BAS
```

Il file batch verrà richiamato dal DOS in questo modo:

```
ARTIFINT ORIG.TXT B:NUOVPROG MIGLPROG
```

Il comando completo, cioè, è formato dal nome del file batch, seguito dai nomi dei file che devono sostituire i parametri fittizi, nell'ordine in cui compaiono nell'elenco di comandi; per separare due diversi valori si usa uno spazio bianco.

In questo esempio i parametri vengono così sostituiti: A:ARTIFINT.BAT va al posto di %0, A:ORIG.TXT al posto di %1, B:NUOVPROG al posto di %2, A:MIGLPROG al posto di %3. In questo modo il file che abbiamo scritto risulta del tutto equivalente a questo:

```
TYPE A:ARTIFINT.BAT
COPY A:ORIG.TXT B:NUOVPROG.TXT
COMP A:ORIG.TXT B:NUOVPROG.TXT
BASICA A:MIGLPROG
```

Se in un file batch bisogna far riferimento ad una specificazione che includa come carattere il simbolo %, questo va ripetuto due volte, per non creare confusione. Se uno dei comandi fosse di visualizzare il directory del file B:TAX%.BAS, il comando da inserire nel file batch dovrebbe essere

```
DIR B:TAX%%.BAS
```

### Il comando SHIFT

Se vi servono più di dieci parametri all'interno dello stesso file, potete usare il comando SHIFT, la cui sintassi è:

SHIFT

Questo comando sposta il valore di ogni parametro a sinistra di una posizione: %0 viene posto uguale al vecchio valore di %1, %1 al vecchio valore di %2 e così via, finché %9 assume il valore dell'undicesimo parametro nella lista dei valori. (Attenzione: il vecchio valore di %0 viene irrimediabilmente perso). Successive esecuzioni di questo comando permettono di accedere a un numero sempre maggiore di parametri.

## VISUALIZZAZIONE DEI COMANDI BATCH: IL COMANDO ECHO

Per mezzo del comando ECHO si può ottenere o evitare la visualizzazione dei comandi di un file batch che vanno in esecuzione. La sintassi è la seguente:

ECHO [ON|OFF|*messaggio*]

Per default lo stato di ECHO è ON, cioè i comandi vengono ripetuti sullo schermo; specificando ECHO OFF si disabilita la visualizzazione; se inve-

ce indicate un messaggio, questo verrà visualizzato, ma senza cambiare lo stato del comando.

## COMANDI PER IL CONTROLLO DELL'ESECUZIONE DEI FILE BATCH: I COMANDI FOR, GOTO E IF

Spesso è utile poter ripetere una sezione di un file batch, o poter controllare alcune condizioni e in qualche caso saltare una parte dei comandi elencati. I comandi DOS per il controllo del flusso operano in modo molto simile ai loro omonimi che controllano il flusso dei programmi BASIC.

Il comando FOR permette di eseguire ripetutamente una serie di comandi di DOS; la sintassi è la seguente:

FOR %%*variabile* IN (*elenco*) DO *comando*

dove %%*variabile* è un nome di variabile, (*elenco*) un elenco di valori e *comando* un qualsiasi comando DOS: ogni successiva iterazione assegna alla %%*variabile* il valore successivo nell'(*elenco*) ed esegue il *comando*; questo permette di cambiare il valore di una variabile ad ogni iterazione del ciclo FOR; purtroppo più istruzioni FOR non possono essere annidate. Come esempio, provate ad eseguire direttamente il seguente comando che visualizzerà prima l'elenco dei file con estensione .BAS e poi quello dei file .BAT:

```
A>FOR %A IN (BAS BAT) DO DIR A:*.%A

A>DIR A:*.BAS

 Volume in drive A has no label
 Directory of  A:\

U15      BAS       128  12-24-84   1:54p
ULTIMO   BAS      2560  12-11-84  12:32a
        2 File(s)    15360 bytes free

A>DIR A:*.BAT

 Volume in drive A has no label
 Directory of  A:\

COPIA    BAT       121   2-13-85  12:39p
AUTOEXEC BAT        23  12-12-84  12:40p
        2 File(s)    15360 bytes free

A>
A>
```

Il comando GOTO permette di trasferire il controllo ad una locazione specifica del file batch; la sintassi è:

GOTO *label*

dove *label* indica la destinazione del salto. L'esecuzione del file batch passerà dalla linea contenente il GOTO a quella che segue immediatamente la linea contenente :*label*. I caratteri significativi in *label* possono essere al massimo otto.

Il comando IF permette di porre delle condizioni per l'esecuzione di alcuni comandi DOS. La sua sintassi è la seguente:

IF [NOT] *condizione comando*

dove il *comando* è eseguito se la *condizione* è verificata (a meno che non venga specificato NOT, nel qual caso il *comando* viene eseguito solo se la *condizione* è falsa).
I parametri che possono essere inclusi in una condizione sono i seguenti:

ERRORLEVEL *numero*
Considerato vero se il programma eseguito precedentemente dal file batch è terminato con un codice d'errore maggiore o uguale al valore indicato.

*stringa1==stringa2*
Considerato vero se le due stringhe sono identiche

EXIST *specfile*
Considerato vero se il file indicato in *specfile* viene trovato nel corretto drive. Non sono consentite indicazioni di cammino in *specfile*.

## IL FILE BATCH AUTOEXEC.BAT

Uno speciale file batch è quello che porta il nome AUTOEXEC.BAT: questo può essere inserito nel disco contenente il DOS e viene eseguito automaticamente ogni volta che il disco è usato per avviare il sistema operativo. Un metodo per creare un nuovo file AUTOEXEC.BAT sul disco che si trova nel drive B: è quello di inviare il seguente comando:

```
COPY CON: B:AUTOEXEC.BAT
```

e quindi di battere tutti i comandi DOS che si vuole inserire in questo file, ad esempio:

```
DATE
BASICA
```

Infine premere F6 ed ENTER per memorizzare il file su disco. D'ora in poi, ogni volta che utilizzerete quel disco per inizializzare il DOS, vi verrà richiesto di inserire la data corrente e verrà avviato l'Advanced BASIC. Anche EDLIN può essere usato per creare e modificare un file batch.

Capitolo

# Introduzione al BASIC

4

In questo capitolo esamineremo l'uso del linguaggio BASIC fornito insieme al DOS 2.10; studieremo in particolare il modo di utilizzarlo per scrivere programmi ed infine come i programmi in BASIC possono essere salvati, caricati ed eseguiti.

## 4.1 Che cos'è il BASIC

Il BASIC è un linguaggio di programmazione ideato per agire come intermediario tra voi e il computer: voi potete informare il PC di quello che intendete fare usando il vocabolario e le regole del BASIC che traduce le istruzioni in una forma direttamente eseguibile dal PC. Sono disponibili altri linguaggi di programmazione che il PC è in grado di utilizzare: il Pascal, il FORTRAN e l'Assembler.
Ognuno di questi linguaggi presenta vantaggi e svantaggi. Il BASIC (*Beginners All-purpose Symbolic Instruction Code*, cioè Codice simbolico generalizzato di programmazione per principianti), ad esempio, è relativamente facile da usare e adatto per un largo spettro di applicazioni, ma se prevedete di utilizzare una grande quantità di programmi sofisticati, sono più convenienti altri linguaggi, come il Pascal.

## 4.2 Le tre versioni del BASIC IBM

Al momento della vendita il PC è dotato di una o più versioni del BASIC IBM: il *Cassette BASIC, il Disk BASIC* e l'*Advanced BASIC*, ciascuna delle quali richiede una diversa configurazione hardware ed ha caratteristiche specifiche. In particolare, queste tre versioni hanno una capacità progressivamente maggiore di memorizzare dati e di eseguire operazioni di input/output. Potete quindi decidere quale versione sia più vicina alle vostre esigenze ed alla configurazione del vostro sistema.
L'IBM fornisce anche, separatamente, una quarta versione del linguaggio, il *Compiled BASIC*, molto simile nell'essenza all'Advanced BASIC, tranne per il fatto che i programmi scritti in questo linguaggio vengono compresi più facilmente dal PC e di conseguenza vengono eseguiti molto più rapidamente. La trattazione del Compiled BASIC va comunque al di là degli scopi di questo libro.

### CASSETTE BASIC

Il Cassette BASIC è memorizzato permanentemente nella memoria ROM (a sola lettura) di ogni PC-IBM: è la versione più semplice del BASIC IBM e, usata con la configurazione minimale, permette di salvare e caricare programmi e dati con l'ausilio di un normale registratore a cassette.
Rispetto alle altre versioni ha minori capacità di manipolazione dell'input e dell'output; mette a disposizione un insieme di 256 caratteri per creare videate e supporta semplici funzioni grafiche (ma solo se possedete la scheda Colore/Grafica); fornisce un controllo rudimentale del sintetizzatore di suoni nell'unità di sistema e consente l'uso di una penna ottica per inviare input.
Il Cassette BASIC funziona su qualunque PC senza bisogno di ampliamenti dell'hardware: non richiede memoria addizionale, ma naturalmente più memoria avete e più grandi possono essere i programmi che fate eseguire (fino ad un massimo di 64K). Potete avviare il Cassette BASIC accendendo il PC oppure operando una reinizializzazione del sistema (premendo i tasti CTRL, ALT e DEL contemporaneamente) quando il sistema è già acceso, purché non siano inseriti dischi nei drive.
Il PC/XT, pur avendo in memoria il Cassette BASIC, non possiede la porta di collegamento per le cassette che si trova nella versione standard: non c'è modo, perciò, di utilizzare memorie di massa con il Cassette BASIC.
Poiché quasi tutti i PC standard usano i dischi per la memorizzazione di dati e programmi, non tratteremo in dettaglio questa prima versione dell'IBM BASIC, ma vi faremo riferimento esplicito solo per evitare confusione con le altre versioni del BASIC.

## DISK BASIC

Il Disk BASIC possiede tutte le caratteristiche del Cassette BASIC integrate da alcune altre molto potenti, la più importante delle quali è la capacità di creare e accedere a informazioni su disco. Altre possibilità sono la comunicazione con dispositivi esterni tramite l'adattatore per comunicazioni asincrone o schede seriali e con al massimo tre stampanti (con il Cassette BASIC potete comunicare con una sola stampante). Inoltre fornisce un orologio che segna l'ora e la data.
Per avviare il Disk BASIC dovete dapprima inizializzare il DOS, come spiegato nel Capitolo 2: ricorderete che ciò viene ottenuto inserendo nel drive A: un disco contenente il DOS ed accendendo il computer o reinizializzando il sistema se questo è già stato acceso. L'esito è la comparsa della videata del DOS:

```
A>keybit

A>wtdatim
Immettere la data odierna (GG-MM-AA): 01-01-1980
Modifica della data
15-03-1985
Immettere l'ora: 00:00:38
Modifica dell'ora
15:30

A>VER

IBM Personal Computer DOS Version  2.10

A>
A>
```

In risposta al prompt caratteristico del DOS (A>) battete:

```
A>BASIC
```

seguito dal tasto ENTER. Questo carica il Disk BASIC dal disco del DOS e fa apparire sul video

```
The IBM Personal Computer Basic
Version D2.10 Copyright IBM Corp. 1981, 1982, 1983
61763 Bytes free
```

### ADVANCED BASIC

Questa versione completa la precedente con alcune caratteristiche: possiede comandi grafici molto potenti e un sofisticato controllo del sintetizzatore di suoni; consente di rilevare eventi esterni, come la pressione di un tasto funzione, dall'interno dell'esecuzione di un programma — comportamento noto sotto il nome di *event trapping* (intercettamento di eventi). Per avviare l'Advanced BASIC, dapprima caricate il DOS e quindi battete il comando:

```
BASICA
```

e vi apparirà la videata dell'Advanced BASIC:

```
The IBM Personal Computer Basic
Version A2.10 Copyright IBM Corp. 1981, 1982, 1983
61327 Bytes free
```

Come avrete notato, tutte le versioni del BASIC visualizzano un numero che corrisponde alla quantità di memoria disponibile per i programmi: questo numero dipende da quanta memoria è installata nel computer, da quanti programmi sono già stati caricati e da quale versione del BASIC state usando. Per il momento ciò non vi deve preoccupare, perché avrete spazio in abbondanza per la maggior parte dei programmi in BASIC: solo quando i vostri programmi diventeranno molti e molto articolati, conoscere il numero di byte disponibili diventerà fondamentale.

## 4.3 Opzioni per l'avvio del BASIC

Sono molte le opzioni che potete richiedere quando avviate sia il Disk BASIC che l'Advanced BASIC: la sintassi completa del comando, infatti, è la seguente:

BASIC[A] ["*specfile*"] [<*stdin*] [[>] >*stdout*] [/F:*nfile*] [/S:*dimbuf*] [/C:*combuf*] [/M: [*max spaziolavoro*] [,*max dimbloc*]] [/D]

Il parametro opzionale *specfile* identifica un file che dovrebbe contenere un programma BASIC: questo programma viene caricato ed eseguito immediatamente sia dal BASIC che dal BASICA; *specfile* può contenere an-

che il drive in cui si trova il programma ed il cammino del directory del file. Per maggiori dettagli su questo parametro fate riferimento al paragrafo 4.8 "Come salvare e caricare programmi".
Le opzioni <*stdin*, >*stdout* e >>*stdout* impongono una redirezione dell'input o dell'output standard, escludendo la tastiera o lo schermo: permettono di ricevere input o di inviare output a dispositivi diversi (come una porta ausiliaria) o ad un file su disco (vedi pag. 89).
Come già visto, il parametro >*stdout* cancella il contenuto del file su disco prima della nuova operazione di scrittura, mentre >>*stdout* permette di aggiungere l'output alla fine del file.
L'opzione /F:*nfile* permette di stabilire il massimo numero di file che possono essere aperti contemporaneamente: se non viene specificata assume il valore di default che è di otto file. In ogni caso non possono essere aperti contemporaneamente più di quindici file. (Vedi Capitolo 7 per l'uso dei file).
L'opzione /S:*dimbuf* consente di dimensionare il buffer che viene utilizzato per i file ad accesso diretto. Il buffer di default è di 128 byte: il valore massimo per questo parametro è 32767. (Vedi Capitolo 7 per maggiori informazioni sull'uso dei file ad accesso diretto).
L'opzione /C:*combuf* dimensiona il buffer da usare per immagazzinare i dati provenienti dall'adattatore per comunicazioni asincrone. Come nel caso precedente il buffer di default è di 128 byte ed il valore massimo è di 32767. (Vedi Capitolo 11 per dettagli sul buffer di comunicazione).
L'opzione /M:*max spaziolavoro* permette di definire il massimo spazio utilizzabile dal BASIC. Può essere usata per riservare spazio di memoria per subroutine in linguaggio macchina o per memorizzazioni speciali. Il valore di default per questa opzione è di 64 Kbyte.
L'opzione *,max dimbloc* permette di riservare spazio sopra il BASIC per caricare programmi scritti in altri linguaggi o in codice macchina.
L'opzione /D impone alle seguenti funzioni BASIC di utilizzare argomenti in doppia precisione: ATN, COS, EXP, LOG, SIN, SQR e TAN. Se non viene specificata, le funzioni utilizzano valori in precisione semplice. (Vedi Capitolo 5 per dettagli sulla precisione numerica e sulle funzioni BASIC).

## 4.4 Come utilizzare la tastiera in BASIC

Nel Capitolo 2 abbiamo descritto la tastiera del PC, ed abbiamo spiegato alcune delle funzioni di editing disponibili dopo l'avvio del BASIC.
Come ricorderete, il tasto BACKSPACE serve per cancellare il carattere immediatamente a sinistra della posizione occupata dal cursore, mentre il tasto ESC cancella l'intera linea; il tasto ENTER viene utilizzato per informare il programma BASIC che la linea è completa e può essere salvata o

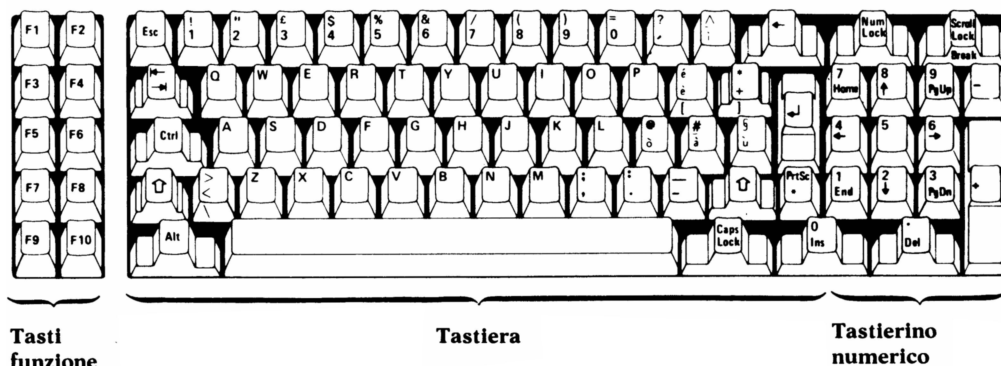



**Figura 4.1**

eseguita. Qui di seguito esamineremo altri tasti che troverete utili nell'uso del BASIC.

## TASTI DI CONTROLLO DEL MOVIMENTO DEL CURSORE

La maggior parte delle funzioni di editing fanno riferimento alla posizione del cursore: per modificarla, solitamente, vengono utilizzati i tasti di controllo del cursore.

**Tabella 4.1** Caratteri speciali BASIC della tastiera del PC

| Carattere | Significato | Carattere | Significato |
|---|---|---|---|
| | Spazio | # | Cancelletto |
| = | Segno di uguale o simbolo di assegnamento | $ | Dollaro |
| | | ! | Punto esclamativo |
| + | Segno più o simbolo di concatenazione | & | "e" commerciale |
| | | , | Virgola |
| – | Segno meno | . | Punto |
| * | Asterisco o segno di moltiplicazione | ' | Apostrofo |
| / | Barra o simbolo di divisione | ; | Punto e virgola |
| \ | Backslash o simbolo di divisione intera | : | Due punti |
| | | ? | Punto interrogativo |
| ^ | Simbolo di esponenziale | < | Minore di |
| ( | Parentesi sinistra | > | Maggiore di |
| ) | Parentesi destra | " | Virgolette |
| % | Simbolo di percentuale | _ | Sottolineatura |

Questi quattro tasti, situati nel tastierino numerico, (tasti numerici 8,2,4 e 6) muovono il cursore di una posizione nella direzione indicata dalla freccia; il movimento verrà ripetuto finché tenete premuto uno di questi tasti.
Nel caso che il cursore si trovi all'estrema sinistra dello schermo e venga premuto il tasto CURSOR LEFT(←), il cursore salterà alla posizione più a destra della linea precedente; se invece si trova all'estrema destra e viene premuto il tasto CURSOR RIGHT(→) si sposterà nella posizione più a sinistra della riga successiva.
Ogniqualvolta vogliate usare i tasti di controllo del cursore dovete assicurarvi che sia disabilitato il modo numerico ed in caso contrario premere il tasto NUM LOCK per disabilitarlo.

### Spostamenti del cursore di parola in parola

Combinando il tasto CTRL con i tasti CURSOR LEFT e CURSOR RIGHT potete muovere il cursore a destra o a sinistra di un'intera parola: una parola è definita come un gruppo di caratteri che inizia con una lettera od un numero ed è separato da altre parole per mezzo di uno spazio o di un carattere speciale. L'elenco dei caratteri speciali si trova nella Tabella 4.1.

## IL TASTO END

Questo tasto serve per spostare il cursore alla fine della linea in cui si trova, come potete vedere nell'esempio:

```
500 PRINT "Inserisci il nome"
```

Il cursore si trova qui

```
500 PRINT "Inserisci il nome"_
```

Il cursore, dopo che il tasto END è stato premuto, si trova qui

## CTRL END

Se tenete premuto il tasto CTRL quando battete il tasto END, viene cancellata la parte di linea che si trova alla destra del cursore, come potete vedere:

```
3290 IF KEY=1 THEN BEEP_ELSE PRINT "Okay"
                       └─ Il cursore si trova qui

3290 IF KEY=1 THEN BEEP _
                        └─I tasti CTRL END lasciano invariata la posizione del cursore ma cancellano il resto della riga
3290 IF KEY=1 THEN BEEP

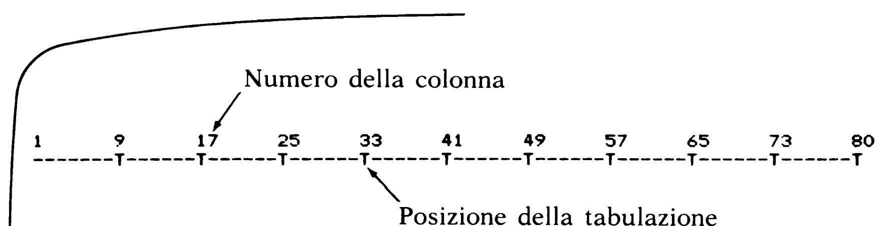
_
└──────────────────────── Premendo il tasto ENTER si memorizza la nuova versione della riga ed il cursore va a capo
```

## IL TASTO HOME

Potete spostare il cursore nell'angolo a sinistra in alto dello schermo in due modi: premendo il tasto HOME (tasto numerico 7) otterrete solo lo spostamento del cursore, mentre premendo contemporaneamente i tasti CTRL e HOME otterrete anche la completa cancellazione dello schermo.

## IL TASTO TAB

Il tasto TAB, indicato con il simbolo →| sulla tastiera, sposta il cursore fino alla successiva tabulazione alla destra della posizione corrente del cursore. Le posizioni di tabulazione ricorrono ogni otto colonne, a partire dal margine sinistro, come mostrato qui di seguito:

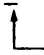



SHIFT TAB non ha alcun effetto.

## IL TASTO DEL

Per cancellare un carattere per volta potete usare il tasto DEL: questo cancella il carattere su cui è posizionato il cursore e sposta il resto della linea di una posizione a sinistra:

```
1200 PRINT "I valori non devono essere interi"
```


Il cursore si trova qui

```
1200 PRINT "I valori on devono essere interi"
```


Il tasto DEL cancella il carattere a cui punta il cursore.

```
1200 PRINT "I valori devono essere interi"
```


Premendo altre tre volte il tasto DEL si ottiene questa linea

Ricordate che il tasto BACKSPACE cancella il carattere a sinistra del cursore, mentre il tasto DEL cancella quello nella stessa posizione del cursore.

## I TASTI FUNZIONE

Due sono i modi in cui vengono usati i tasti funzione in BASIC. Innanzitutto essi possono essere usati come tasti definibili dall'utente. In questo modo, quando premete un tasto funzione viene visualizzata una parola o un comando specifico, cioè potete scrivere un'intera parola od un comando battendo un unico tasto.
Il secondo scopo di un tasto funzione è quello di interrompere l'esecuzione di un programma, azione possibile, però, solo in Advanced BASIC. Nel Capitolo 6 verrà approfonditamente spiegato l'uso dei tasti funzione come tasti di interruzione.
Quando viene avviato il BASIC i tasti funzione vengono immediatamente inizializzati come tasti definibili e ad ogni tasto viene assegnato un comando specifico. Le prime lettere della parola o del comando appaiono sull'ultima linea dello schermo e rimangono finché voi (da tastiera o da programma) non le cancellate. Inoltre, se ridefinite i tasti funzione, appariranno nella linea le nuove assegnazioni.
Il numero visualizzato a sinistra del comando indica il numero del corri-

spondente tasto funzione: se avete predisposto lo schermo a 40 colonne, verranno visualizzate solo le funzioni da F1 a F5, mentre appariranno tutte se lo schermo è a 80 colonne.
La maggior parte delle stringhe che vengono assegnate ai tasti funzione corrispondono ai comandi più comuni in BASIC per lo sviluppo dei programmi, come i comandi LOAD e RUN che vengono usati ogni volta che volete caricare un programma in memoria e farlo eseguire.
In alcune di queste stringhe è compreso il codice corrispondente al tasto ENTER in modo che premere, ad esempio, il tasto F2 corrisponde a battere il comando RUN e premere poi il tasto ENTER. I tasti funzione che comprendono il codice di ENTER sono seguiti da una freccia (←) ad eccezione dei tasti F6 ed F10, in cui è compreso ENTER, ma non viene visualizzato sullo schermo.
Per ottenere l'elenco del significato in BASIC dei tasti funzione battete il comando:

```
KEY LIST
```

ed avrete in risposta:

```
Ok
KEY LIST
F1 LIST
F2 RUN
F3 LOAD"
F4 SAVE"
F5 CONT
F6 ,"LPT1:"
F7 TRON
F8 TROFF
F9 KEY
F10 SCREEN 0,0,0
```

Nel seguito di questo capitolo impareremo ad usare i tasti funzione F1 (LIST), F2 (RUN), F3 (LOAD), F4 (SAVE). Nel Capitolo 6, invece, vi mostreremo come ridefinire i tasti funzione per mezzo del BASIC.

## IL TASTO ALT

Il tasto ALT consente di generare parole chiave del BASIC e codici ASCII dei caratteri. Se, tenendo premuto il tasto ALT battete un tasto alfabetico, visualizzate sullo schermo una parola chiave del BASIC: le parole chiave sono quei comandi e quelle istruzioni che vi servono per scrivere i programmi in BASIC. In Tabella 4.2 trovate le corrispondenze tra tasti alfabetici e parole chiave.

**Tabella 4.2** Corrispondenze tra tasti alfabetici e parole chiave

| ALT | Parole chiave | ALT | Parole chiave | ALT | Parole chiave |
|---|---|---|---|---|---|
| A | AUTO | J | (nessuna) | S | SCREEN |
| B | BSAVE | K | KEY | T | THEN |
| C | COLOR | L | LOCATE | U | USING |
| D | DELETE | M | MOTOR | V | VAL |
| E | ELSE | N | NEXT | W | WIDTH |
| F | FOR | O | OPEN | X | XOR |
| G | GOTO | P | PRINT | Y | (nessuna) |
| H | HEX$ | Q | (nessuna) | Z | (nessuna) |
| I | INPUT | R | RUN | | |

Se invece, sempre tenendo premuto il tasto ALT, battete un numero di tre cifre usando il tastierino numerico, inviate al PC il codice ASCII di uno dei caratteri elencati nell'Appendice B, possibilità che si rivela molto utile nel caso in cui vogliate visualizzare uno dei caratteri speciali che non trovate nella tastiera del PC.

## 4.5 I due modi di funzionamento del BASIC

Il BASIC può essere utilizzato sul PC in due modi: quello chiamato *modo diretto* e quello chiamato *modo indiretto* o *programmato*. Nel modo diretto il BASIC dà una risposta immediata ad ogni istruzione o comando che voi inviate, cioè il PC si comporta in un certo senso come una calcolatrice. Il modo indiretto, invece, viene usato quando intendete conservare le istruzioni per un uso successivo, cioè quando scrivete programmi. Entrambi i modi funzionano con tutte e tre le versioni del BASIC.

### USO DEL MODO DIRETTO

Il modo diretto viene usato per eseguire velocemente dei calcoli o per controllare l'esecuzione di un'istruzione BASIC.
Questo modo possiede due importanti caratteristiche: la prima è che il BASIC dà una risposta immediata ad un comando; la seconda è che quando viene eseguito un comando, questo non rimane nella memoria del PC e non fa parte di alcun programma BASIC anche se appare ancora sullo schermo. Potete naturalmente mandare in esecuzione una seconda volta lo stesso comando semplicemente riportando il cursore sulla linea che

contiene il comando in questione e battendo di nuovo il tasto ENTER.
Per esercitarvi con il modo diretto, provate a calcolare il valore di 512/16; in risposta al prompt battete la linea seguente, seguita dal tasto ENTER:

```
PRINT 512/16
```

Dopo aver premuto ENTER il PC vi risponderà in questo modo:

```
Ok
PRINT 512/16
 32
Ok
```

Questo comando è composto da due parti: la prima è rappresentata dalla parola chiave PRINT, che è un comando (o istruzione) BASIC e significa "visualizza tutto ciò che appare alla destra di questo comando". Se alla destra si trova un'espressione algebrica, come quella del nostro esempio, ne viene visualizzato il risultato; se invece si trova una stringa, cioè una sequenza di caratteri racchiusi tra virgolette, viene visualizzato il testo tra virgolette.
La seconda parte di questa linea in modo diretto è l'espressione 512/16: il BASIC riconosce il simbolo / come l'operatore di divisione.
Per osservare la visualizzazione di stringhe di testo, eseguite il seguente comando:

```
PRINT "Questo e' il modo diretto"
```

seguito dal tasto ENTER. Il PC visualizza:

```
Ok
PRINT "Questo e' il modo diretto"
Questo e' il modo diretto
Ok
```

Quando riportate il cursore sulla linea che contiene un comando eseguito in precedenza potete fare alcuni cambiamenti prima di mandarla di nuovo in esecuzione. Lo schermo del PC potrebbe apparire, per esempio, come nella figura seguente.

```
Ok
PRINT 512/16
 32
Ok
PRINT "Questo e' il modo diretto"
Questo e' il modo diretto
Ok
_
```
Il cursore si trova qui

Riportate il cursore sulla riga contenente il primo comando per mezzo dei tasti con le frecce che trovate nel tastierino numerico; ricordate che se compaiono dei numeri al posto dei movimenti del cursore, dovete uscire dal modo numerico premendo il tasto NUM LOCK.

```
Ok
PRINT 512/16
 32
Ok
PRINT "Questo e' il modo diretto"
```
Premete i tasti CURSOR UP e CURSOR RIGHT per spostare il cursore in questa posizione

Ora correggete il numero battendo 64 al posto di 16 e poi premete il tasto ENTER: il PC eseguirà il nuovo calcolo in modo diretto, come segue:

```
Ok
PRINT 512/64
 8
Ok
PRINT "Questo e' il modo diretto"
```
Il cursore ora si trova qui

Poiché una linea di comando in modo diretto non viene mai salvata in memoria, risulta definitivamente persa quando viene cancellata dallo schermo; potete in ogni modo memorizzare informazioni anche in modo diretto con l'ausilio di variabili. Nel Capitolo 5 spiegheremo diffusamente il concetto di variabile in BASIC: per il momento potete considerare una variabile come una parola o una lettera di vostra scelta a cui asse-

gnate un valore e a cui potete riferirvi anche successivamente. Inviate il seguente comando in modo diretto:

```
PIANO=88
```

per memorizzare il numero 88 sotto l'"etichetta" PIANO. Provate poi a far visualizzare il valore della variabile per mezzo del comando PRINT.

```
Ok
PIANO = 88
Ok
PRINT PIANO
 88
Ok
```

Il BASIC va a cercare il valore corrente della variabile PIANO e lo visualizza.
Fino ad ora abbiamo inviato un comando per volta in modo diretto, ma potete anche inviare una serie di comandi in un'unica linea, purché ogni comando sia separato dal successivo da un due punti. Provate a battere le due righe seguenti di comandi:

```
FOR X=1 TO 10:FOR Y=1 TO X:PRINT " ";:
NEXT Y:PRINT "Riga numero " X:NEXT X
```

Dopo che avete premuto ENTER il PC vi risponderà:

```
Ok
FOR X=1 TO 10:FOR Y=1 TO X:PRINT " ";:
NEXT Y:PRINT "Riga numero " X:NEXT X
 Riga numero  1
  Riga numero  2
   Riga numero  3
    Riga numero  4
     Riga numero  5
      Riga numero  6
       Riga numero  7
        Riga numero  8
         Riga numero  9
          Riga numero  10
Ok
```

Se vi appare il messaggio SYNTAX ERROR, significa che probabilmente avete commesso un errore nel battere la linea. Nel Capitolo 5 esamineremo tutte le istruzioni che compongono questo comando in modo diretto.

## USO DEL MODO INDIRETTO

Finora, usando il modo diretto, avete scritto e fatto eseguire una linea per volta, contenente uno o più comandi BASIC. Nel modo indiretto, invece, potete scrivere una serie di linee che vengono memorizzate ed utilizzate in seguito per l'esecuzione: questa successione di linee viene chiamata *programma*. Il modo indiretto viene perciò usato per creare programmi BASIC.
Anche nel modo indiretto troviamo due caratteristiche distintive: la prima è che in questo modo possono essere utilizzate diverse linee di istruzioni per risolvere un solo problema e queste linee possono essere memorizzate; la seconda è che il modo indiretto viene utilizzato automaticamente ogniqualvolta iniziate una linea in BASIC con un numero detto *numero di linea*. Quando poi premete ENTER al termine della linea in BASIC, questa diventa parte del programma corrente che viene salvato nella memoria interna del PC. Se vi è necessario, potete visualizzare nuovamente questa linea richiamandone il numero con il comando LIST.
Scriviamo ora un breve programma in modo indiretto: in risposta al prompt battete le righe seguenti, terminandole con il tasto ENTER.

```
Ok
NEW
Ok
10 PRINT "Questo e' un utile programma"
20 BEEP
30 GOTO 10
```

Il comando NEW, che avete inviato in modo diretto, cancella dalla memoria ogni eventuale programma precedente.
Prima di passare alla fase esecutiva, vi ricordiamo che potete in ogni momento interrompere l'azione del PC premendo contemporaneamente i tasti CTRL e BREAK. A questo punto mandate in esecuzione il programma, premendo il tasto F2, che equivale a scrivere RUN seguito da ENTER.
Il PC risponde con:

```
Ok
RUN
Questo e' un utile programma (beep)
Questo e' un utile programma (beep)
Questo e' un utile programma   .
Questo e' un utile programma   .
Questo e' un utile programma   .
```

Notate che se interrompete il programma con CTRL BREAK dopo la scritta BREAK compare il numero dell'ultima linea di programma eseguita dal PC.
Abbiamo visto come funziona il tasto F2, cioè il comando RUN; proviamo ora il comando LIST, premendo F1 seguito da ENTER. Il PC risponde visualizzando il breve programma che abbiamo composto.

```
Ok
LIST
10 PRINT "Questo e' un utile programma"
20 BEEP
30 GOTO 10
```

Nel resto di questo capitolo focalizzeremo la nostra attenzione sull'uso del modo indiretto, cioè sull'arte di scrivere programmi. Esaminiamo perciò più da vicino come vengono costruiti i programmi.

## 4.6 Linee di programma

I programmi in BASIC hanno il loro fondamento nel concetto di linea; è necessario perciò precisare che esistono due tipi di linea: la linea "fisica" e la linea "logica".
Una linea "fisica" è semplicemente una linea (o una riga) del dispositivo di output che state utilizzando; se questo è lo schermo, una linea fisica è composta da 40 o da 80 caratteri, a seconda della larghezza che avete scelto.
Una linea "logica", invece, viene misurata in modo differente: può essere lunga fino a 255 caratteri e termina con il carattere corrispondente a ENTER. Questo è il tipo di linea più importante, al momento, poiché rappresenta l'unità di informazione elaborata dal BASIC.

Una linea logica del BASIC può essere composta da più linee fisiche visualizzate. Quello che conta per stabilirne la fine è il punto in cui è stato premuto il tasto ENTER e non il termine della linea sullo schermo.
Nel seguente esempio potete vedere come una linea BASIC sia composta da più linee fisiche:

```
230 PRINT "Divampa una lotta scabrosa,  ora sottile
ora sfacciata, che viene riferita da un osservatore
non parziale, anzi giustamente orientato nei propri
giudizi. (G.Greene)"
```

Se però cercate di scrivere una linea BASIC (logica) composta da un numero di caratteri maggiore di 255, quelli in sovrappiù andranno persi non appena battete ENTER.

## NUMERI DI LINEA

Un numero che preceda una linea logica viene chiamato numero di linea: come abbiamo visto, quando assegnate un numero ad una linea, questa viene memorizzata e non viene eseguita immediatamente. Il numero di linea funge da etichetta per la linea in memoria che viene a far parte del corrente programma BASIC.
Un numero di linea deve sempre occupare la prima posizione di una linea logica (ma naturalmente può anche non coincidere con l'inizio di una linea fisica) e può essere compreso tra 0 e 65535.
Ecco alcuni esempi di uso corretto dei numeri di linea:

```
1010 PRINT "Questo e' un numero di linea grande"
65000 GOTO 65100
```

Il numero di linea segna dunque l'inizio della linea BASIC: questa deve essere separata da esso da almeno uno spazio bianco.

### Come numerare le linee di programma

Nella prima stesura di un programma BASIC è buona norma numerare le linee di dieci in dieci: questo rende più facile l'inserimento di eventuali linee addizionali. Potreste iniziare un programma in questo modo:

```
10 ' Ecco un buon sistema per numerare le linee di
programma
20 GOTO 500 ' Inizializzazione del sistema
30 GOTO 600 ' Input dell'operatore
```

## I comandi AUTO e RENUM

Vi risulterà più semplice numerare le linee dei vostri programmi se utilizzerete i comandi AUTO e RENUM. Se inviate il comando AUTO, il PC inizierà a generare numeri di linea ed ogni volta che premerete il tasto ENTER apparirà un nuovo numero sulla successiva riga dello schermo.
La sintassi di questo comando è:

AUTO [*linea*] [,[*incremento*]]

Il parametro opzionale *linea* indica il primo numero di linea che il BASIC genererà; il parametro *incremento*, invece, specifica l'incremento tra un numero e il successivo. Se entrambi i parametri vengono omessi, sia il primo numero che l'incremento assumono il valore di default 10. Se perciò battete il comando:

```
AUTO
```

i numeri di linea saranno:

```
10 (linea BASIC)
20 (linea BASIC)
30 (linea BASIC)
      .
      .
      .
```

Se volete uscire dal modo AUTO, premete i tasti CTRL BREAK: il PC vi risponderà con il prompt Ok, il che significa che siete di nuovo in modo diretto. Ponete attenzione al fatto che se usate i tasti CTRL BREAK la linea che state scrivendo non verrà memorizzata come parte del programma corrente.
Se vi occorrerà inserire una nuova linea tra due successive potete usare il comando RENUM; supponete di voler apportare alcune correzioni al seguente programma:

```
10 PRINT "Questa e' una lista di valori di x e di y"
15 FOR x=0 TO 70
16 y=x*5
20 PRINT x,y
30 NEXT
40 END
```

e di voler tra l'altro inserire una linea tra la 15 e la 16. Dovrete, come prima operazione, rinumerare le linee già esistenti per mezzo del comando

```
RENUM
```

che assegnerà numeri di linea con incremento pari a 10. Siete ora in grado di inserire fino a nove linee tra le originali 15 e 16 che sono divenute la 20 e la 30.
Affronteremo altri problemi di editing di linee di programma nel prossimo paragrafo; inoltre, maggiori dettagli di questi due comandi, AUTO e RENUM, sono riportati nell'Appendice A.

## CONTENUTO DI UNA LINEA BASIC

Ogni linea BASIC (cioè ogni linea di programma) contiene una o più "frasi" BASIC, dette in alcuni casi anche *comandi*: una frase BASIC è formata da istruzioni eseguibili dal computer, dati che possono essere utilizzati in altre istruzioni o commenti che aiutano ad interpretare le intenzioni del programmatore.

### Istruzioni BASIC

Un'istruzione BASIC è composta da una successione di *parole chiave* e, se necessario, da un *argomento*. Prendiamo come esempio le linee seguenti:

```
300 PRINT "Questa parte tra virgolette e'l'argomento
della parola chiave -PRINT- "
310 IF x=4 THEN 450
```

Nella linea 300, la parola chiave è PRINT, cioè l'istruzione BASIC PRINT e l'argomento è la stringa che segue la parola. Nella linea 310, invece, le parole chiave sono IF e THEN, cioè l'istruzione BASIC IF-THEN e gli argomenti sono X=4, cioè un'espressione, e 450, che in questo caso si riferisce ad un'altra linea di programma.

### Istruzioni multiple

Le linee di programma BASIC possono essere formate da più di un'istruzione: in questo caso ogni istruzione dev'essere separata dalla successiva per mezzo di un segno di due punti, come potete vedere:

```
600 FOR x=1 TO 500:NEXT x
```

In questo esempio, la prima istruzione è un'istruzione FOR, con argomenti X, 1 e 500 e la seconda è un'istruzione NEXT. Quando viene eseguita questa linea di programma, il processo viene ripetuto per 500 volte: è questo un buon mezzo per generare ritardi nei programmi.

### Spazi

Gli spazi bianchi hanno differente valore in BASIC, a seconda del punto in cui compaiono nella linea: uno spazio è obbligatorio in alcuni punti, come tra numero di linea e parola chiave.
In generale le parole chiave devono essere separate di almeno uno spazio vuoto da qualsiasi altra cosa nella linea; un'eccezione a questa regola si presenta quando la parola segue immediatamente un due punti, che in questo caso funge da separatore esattamente come lo spazio. I separatori in BASIC servono per suddividere le varie parti della linea.
Nella maggior parte degli altri casi il BASIC ignora gli spazi bianchi, a meno che non facciano parte di una stringa.

### Lettere maiuscole e minuscole

In generale il BASIC non fa distinzione tra maiuscole e minuscole, nel senso che converte automaticamente tutte le parole chiave e i nomi di variabili in lettere maiuscole. Solo quando i caratteri fanno parte di una stringa (cioè sono racchiusi tra virgolette) o di un commento (cioè seguono la parola REM o un apostrofo) vengono conservate anche le lettere minuscole.

## 4.7 Comandi per la stesura dei programmi

Vi sono alcuni comandi BASIC che facilitano la memorizzazione di nuove linee di programma e la loro modifica quando necessario.

### COME PARTIRE DA ZERO: IL COMANDO NEW

Ogni linea BASIC che scrivete viene memorizzata come parte del programma corrente. Se volete iniziare un nuovo programma dovete prima inviare il comando:

```
NEW
```

che cancella tutte le variabili e ogni linea del vecchio programma dalla memoria del PC.

## COME CANCELLARE LO SCHERMO: IL COMANDO CLS

Se, quando volete scrivere un nuovo programma, lo schermo è pieno di informazioni inutili, potete cancellarlo completamente e riportare il cursore nell'angolo in alto a sinistra, detto posizione "home" per mezzo del comando:

```
CLS
```

Otterrete lo stesso effetto premendo il tasto HOME contemporaneamente al tasto CTRL.

## COME CREARE DEI PROGRAMMI

Per generare un programma non dovete far altro che battere le singole linee di programma e memorizzarle premendo il tasto ENTER.
Se vi accade di commettere errori, potete, usando gli appositi tasti di controllo, riportare il cursore sull'errore e quindi correggere. Nel caso in cui dobbiate sostituire un solo carattere all'interno di una linea, portate il cursore sul carattere da modificare e battete direttamente il nuovo carattere sopra il vecchio, come potete vedere nella sequenza qui riportata:

```
1010 IF GRETA=GAble THEN 320 ELSE 410_
```

Il cursore è qui

```
1010 IF GRETA=GAble THEN 320 ELSE 410
```

Il tasto CURSOR LEFT sposta il cursore in questa posizione

```
1010 IF GRETA=GARBO_THEN 320 ELSE 410
```

Il nuovo testo è ribattuto sopra il vecchio e il cursore si sposta

```
1010 IF GRETA=GARBO THEN 320 ELSE 410
_
```

Premendo il tasto ENTER si memorizza la nuova versione della linea ed il cursore si muove alla linea seguente

## IL MODO INSERT

Se dovete aggiungere una parte di testo nel mezzo di una linea, potete entrare in *modo Insert* premendo il tasto INS. Riconoscerete di essere in questo nuovo modo perché il cursore raddoppierà le proprie dimensioni. Per uscire dal modo Insert potete premere ENTER, uno dei tasti di controllo del cursore o di nuovo il tasto INS.
Una volta entrati nel modo Insert iniziate a scrivere e noterete che tutti i caratteri che si trovano alla destra di quelli che state inserendo verranno progressivamente spostati ancora più a destra. Se la linea viene ad occupare anche la riga di schermo successiva tutte le linee seguenti saranno di conseguenza abbassate di una linea. Nell'esempio seguente potete osservare l'uso del modo Insert:

```
3000 PRINT "ARRIVA PRIMAVERA"_
```

Il cursore si trova inizialmente qui

```
3000 PRINT "ARRIVA PRIMAVERA"
```



Il cursore viene spostato in questa posizione e viene premuto il tasto INS; il cursore diventa più grande

```
3000 PRINT "ARRIVA LAPRIMAVERA"
```



Il cursore si sposta man mano che viene inserito nuovo testo; tutto ciò che si trova alla sua destra viene spostato di conseguenza

```
3000 PRINT "ARRIVA LA PRIMAVERA"
```



Il cursore ora è qui

```
3000 PRINT "ARRIVA LA PRIMAVERA"
```



Dopo aver premuto il tasto ENTER il cursore si trova nella linea successiva e termina il modo Insert

## COME LISTARE UN PROGRAMMA: IL COMANDO LIST

Potete richiamare su uno qualsiasi dei dispositivi del PC la versione del programma in memoria per mezzo del comando LIST; è una possibilità questa che vi risulterà molto utile soprattutto nella fase di scrittura di un programma:

```
Ok
LIST
10 'Questo e' un esempio notevole
20 'di come listare un programma
30 BEEP
Ok
```

In alternativa al comando LIST potete naturalmente usare il tasto funzione F1 seguito dal tasto ENTER; inoltre, per evitare di creare confusione, sarà opportuno inviare dapprima il comando CLS per cancellare ogni eventuale informazione dallo schermo.
La sintassi completa del comando LIST è la seguente:

LIST [*linea1*] [–[*linea2*]] [,"*specfile*"]

Il primo parametro indica il numero di linea da cui deve partire il listato ed il secondo il numero dell'ultima linea da listare. Se non viene specificato alcun parametro, sarà listato l'intero programma.
Se invece indicate un unico numero nel comando LIST, verrà visualizzata quell'unica linea, come potete vedere:

```
Ok
LIST 10
10 'Questo e' un esempio notevole
Ok
```

Utilizzando un punto (.) come parametro si ottiene la visualizzazione della linea corrente, cioè dell'ultima linea visualizzata o memorizzata. Questo procedimento può essere utile qualora vi servisse verificare una linea che avete appena scritto. Ricordate che il punto può essere utilizzato per indicare la linea corrente in ogni comando BASIC in cui l'argomento sia un numero di linea.

Se specificate solo il primo parametro ed il segno –, potete far iniziare il listato dalla linea scelta fino alla fine del programma. Un esempio potrebbe essere il seguente:

```
Ok
LIST 20-
20 'di come listare un programma
30 BEEP
Ok
```

Se invece volete listare il programma dall'inizio fino ad una determinata linea, tralasciate il primo parametro ed indicate il segno – ed il secondo.

```
Ok
LIST -20
10 'Questo e' un esempio notevole
20 'di come listare un programma
Ok
```

*specfile*, l'ultimo parametro da inserire, indica la destinazione del listato del programma: se viene tralasciato, questo appare sullo schermo, ma se vi interessa conservarlo in forma leggibile potete specificare come destinazione un file su disco o su stampante. Se volete conservare un listato del programma TOPAZ, che ora avete in memoria, nel disco inserito nel drive B:, inviate il seguente comando:

```
LIST ,"B:TOPAZ.LST"
```

Fate attenzione all'estensione del nome del file: .LST vi ricorda che il file contiene solo un listato del programma e non il programma stesso. Presto tratteremo il modo di inviare ad una stampante il listato di un programma.

**Come sospendere temporaneamente la visualizzazione del listato**

Se il listato che sta comparendo sul vostro schermo è più lungo di una videata, potete fermarne temporaneamente la visualizzazione premendo contemporaneamente i tasti CTRL e NUM LOCK. Il listato ripartirà non appena premerete un qualsiasi tasto (tranne SHIFT, BREAK o INS).

### Come interrompere un listato

Se state visualizzando il listato di un programma e volete interromperlo (per scrivere una nuova parte di programma o altro...) premete il tasto BREAK contemporaneamente al tasto CTRL.
Se invece il listato è diretto ad un altro dispositivo, come una stampante, non potete interromperlo.

### Come stampare un listato

Se avete a disposizione una stampante potete produrre una copia su carta del vostro listato per mezzo del comando:

```
LIST ,"LPT1:"
```

L'intera sintassi sarebbe:

LIST [*linea1*] [–[*linea2*]], "LPT1:"

dove i parametri *linea1* e *linea2* hanno esattamente le stesse funzioni di quelli visti precedentemente per il video. Invece di battere la sigla LPT1: potete semplicemente premere il tasto funzione F6, che contiene anche il codice ENTER. Insomma, per avere una stampa del listato completo del programma residente in memoria potete semplicemente premere il tasto F1 (per LIST) seguito dal tasto F6.
Per avere una copia stampata del vostro listato potete seguire anche un altro metodo, cioè usare il comando LLIST, la cui sintassi è:

LLIST [*linea1*] [–[*linea2*]]

In questo modo il comando:

```
LLIST 50-
```

risulta identico a:

```
LIST 50- ,"LPT1:"
```

## COME AGGIUNGERE LINEE DI PROGRAMMA

Per aggiungere nuove linee di programma non dovete fare altro che battere un nuovo numero di linea seguito dal testo della linea stessa: quando questa è completa premete ENTER per memorizzarla.

## COME CAMBIARE LINEE DI PROGRAMMA

Per correggere un programma dovete dapprima visualizzare il tutto o la parte che vi interessa sullo schermo. Come avete già visto ciò si ottiene per mezzo del comando LIST.
Una volta che la linea di programma che volete modificare è stata visualizzata, usate i soliti tasti di controllo cursore per portare il cursore nel punto che vi interessa modificare: a questo punto potete riscrivere la nuova versione sopra la vecchia oppure aggiungere una parte di testo in modo Insert. Quando avrete terminato l'operazione, premete il tasto ENTER avendo l'accortezza di controllare che il cursore si trovi ancora nella linea corretta. Il BASIC sostituirà nel programma alla vecchia linea quella che avete appena terminato di scrivere.
Se la linea in questione richiede così tanti cambiamenti da far risultare scomoda la correzione pezzo per pezzo, può essere più conveniente riscriverla per intero: in questo caso ponete il cursore su di una riga vuota dello schermo, battete qui la nuova versione e premete ENTER quando avete terminato; la nuova linea prenderà automaticamente il posto della vecchia, anche nel caso in cui la vecchia versione appaia ancora sullo schermo.
In alternativa potete usare il tasto ESC per cancellare l'intera linea logica, ma prima di iniziare l'operazione dovete ricordarvi di riscrivere il numero della linea interessata. Ponete attenzione al fatto che il tasto ESC cancella la linea logica solamente dallo schermo e non dalla memoria, come potete notare nel seguente esempio:

```
870 GOTO 500
880 ON ERROR GOTO 3000_
890 BEEP
Ok
```

Il cursore si trova in questa posizione

```
870 GOTO 500
_
890 BEEP
Ok
```

Il cursore torna all'inizio della riga quando si preme ESC

```
870 GOTO 500
LIST 870-890_
890 BEEP
Ok
```

Quando viene scritto il comando LIST il cursore si sposta di conseguenza; il tasto ENTER manda in esecuzione il comando a partire dalla riga di schermo seguente

```
870 GOTO 500
LIST 870-890
870 GOTO 500
880 ON ERROR GOTO 3000
890 BEEP
Ok
```

Notate che la linea cancellata dal tasto ESC (la 880) fa ancora parte del programma

**COME CANCELLARE LINEE DI PROGRAMMA**

Per cancellare un'intera linea di programma, dovete batterne il numero su di una riga di schermo totalmente vuota e premere subito il tasto ENTER. Il BASIC cancellerà questa linea dal programma poiché la nuova versione non contiene assolutamente nulla. Osservate l'esempio seguente:

```
LIST
10 'Questo e' un breve programma
20 PRINT "La vostra altezza e' -" ALTEZZA
30 GOTO 20
40 END
Ok
30
LIST
10 'Questo e' un breve programma
20 PRINT "La vostra altezza e' -" ALTEZZA
40 END
Ok
```

Per cancellare un gruppo di linee consecutive potete usare il comando DELETE, con la sintassi:

DELETE [*linea1*] [–*linea2*]

oppure

DELETE [*linea1*–]

I parametri *linea1* e *linea2* specificano l'intervallo in cui cancellare le linee: se viene tralasciato il primo parametro, l'operazione partirà dalla prima linea in memoria, se viene tralasciato il secondo, verrà cancellata solo la linea indicata. La seconda sintassi, invece, permette di cancellare

tutte le linee da quella indicata fino alla fine del programma; il punto (.) può essere usato per indicare la linea corrente. Il comando:

```
DELETE .-
```

serve perciò per cancellare tutte le linee da quella corrente fino alla fine del programma.

### COME DUPLICARE UNA LINEA DI PROGRAMMA

Per creare il duplicato di una linea, cioè una linea identica ad un'altra tranne che per il numero di linea, visualizzate la vecchia linea e cambiatene il numero ribattendone sopra uno nuovo, infine premete ENTER. La stessa linea logica ora compare in due diversi punti del programma, corrispondenti al nuovo e al vecchio numero di linea: notate che questa operazione non cancella la vecchia linea di programma.
La tecnica qui descritta risulta molto utile anche nel caso in cui vi occorra aggiungere una nuova linea di programma simile ad un'altra già esistente in memoria. In questo caso, visualizzate la vecchia linea con il comando LIST, apportate le variazioni necessarie e cambiate il numero di linea; infine, come sempre, premete il tasto ENTER e la nuova linea verrà a far parte del programma.

## 4.8 Come salvare e caricare programmi

Dopo aver creato un programma BASIC ed averlo memorizzato, potete salvarlo su disco: questo vi permette di caricarlo di nuovo in memoria ogni volta che sia necessario.

### SPECIFICAZIONE DI FILE

Quando un comando BASIC si riferisce esplicitamente ad un file su disco o ad altri dispositivi particolari, dovete identificare in modo esatto il file che deve essere usato; questa identificazione è permessa dalla specificazione del file, nella forma:

[*dispositivo*:] [*cammino* \ ] *nomefile*

Il parametro facoltativo *dispositivo*: indica da o su quale drive o altro dispositivo il file debba essere letto o scritto. I nomi di dispositivo validi

comprendono, tra i più frequenti: KYBD: (tastiera, solo per input), SCRN: (schermo, solo per output), LPT1:, LPT2:, LPT3: (stampanti, per output o accesso diretto), COM1:, COM2: (interfaccia per comunicazione asincrona, sia per input che per output) e A:, B:, C:, D: (drive per dischi, sia input che output). Se in una specificazione non viene indicato il dispositivo, viene utilizzato per default il drive corrente sia per operazioni di lettura che di scrittura.
Il *nomefile*, per un file su disco, ha la seguente sintassi:

*nome.est*

Il parametro *nome* può essere lungo da uno a otto caratteri ed essere composto da qualsiasi lettera o cifra decimale o dai seguenti segni particolari:

( ) { }
@ # $ % & !
- _ ` ' ~ ¦

L'estensione *est*, opzionale, può essere composta al massimo da tre caratteri, scelti nello stesso insieme definito prima per i nomi.
Il BASIC inserisce un punto dopo l'ottavo carattere se il parametro *nome* è più lungo di otto caratteri e non avete indicato un'estensione ed usa i primi tre caratteri successivi come estensione. Se l'estensione da voi indicata è più lunga di tre caratteri, vengono conservati solo i primi tre.
Negli esempi seguenti potete vedere come il BASIC tratta i nomi dei file:

| **Nome digitato** | **Nome interpretato dal BASIC** |
|---|---|
| PROGROT.BAS | PROGROT.BAS |
| PROGROTBAS | PROGROTB.AS |
| PROGROTBASAA | PROGROTB.ASA |
| PROGROTAA.BAS | nome illegale – troppi caratteri |

Poiché i programmi BASIC possono avere accesso ai vari file immagazzinati su disco, anche il BASIC dev'essere in grado di supportare la struttura ad albero dei directory caratteristica del DOS; in questo modo dovete poter indicare un cammino per ogni file, cioè un elenco di directory, separati l'uno dall'altro per mezzo di backslash. (Vedi Capitolo 3 per ulteriori informazioni sulla struttura ad albero). Un esempio di specificazione che include un cammino potrebbe essere:

```
B:\STUDENTI\ANALISI1\ROSSI.TXT
```

Se non è indicato alcun cammino, si assume che il file si trovi nel directory corrente, cioè in quello in cui vi trovate quando avviate il BASIC. Conoscete già tre comandi che vi permettono di manipolare i directory: MKDIR (per creare un nuovo directory), CHDIR (per cambiare il directory corrente) e RMDIR (per cancellare un directory vuoto): i comandi BASIC con questi nomi si comportano esattamente come gli omonimi comandi DOS discussi nel Capitolo 3. (Vedi anche Appendice A per dettagli sui comandi relativi al directory). Vi mostriamo ora come creare il nuovo directory \NUOVODIR sul drive B: e memorizzare il programma BASIC corrente come file PROG di questo directory.

```
MKDIR "B:\NUOVODIR"
SAVE "B:\NUOVODIR\PROG"
```

I comandi elencati qui di seguito sono quelli che permettono l'uso di una specificazione in cui sono indicati l'identificatore del dispositivo, un cammino ed un nome di file:

| | | | | |
|---|---|---|---|---|
| BLOAD | BSAVE | CHAIN | CHDIR | FILES |
| KILL | LOAD | MERGE | MKDIR | NAME |
| OPEN | RMDIR | RUN | SAVE | |

**COME SALVARE I PROGRAMMI**

Prima di salvare un programma dovete preparare un disco formattato (come descritto nel Capitolo 3) ed inserirlo in un drive; potete poi usare il comando SAVE, la cui sintassi è la seguente:

SAVE *"specfile"* [,A]

oppure

SAVE *"specfile"* [,P]

L'opzione P consente di "proteggere" il vostro programma nel senso che dopo essere stato salvato con l'opzione P può solo essere caricato ed eseguito: non potete più listarlo, né con il comando LIST, né con LLIST. Prima di usare l'opzione P, perciò, assicuratevi di aver memorizzato una copia non protetta del programma per ogni eventuale necessità che si presenti più avanti.
L'opzione A permette di salvare il programma in formato ASCII (o *text*): questo vi consente di correggere il programma con un word processor se lo ritenete più facile. Vi consente anche di fondere il vostro programma

in un altro programma BASIC (vedi comando MERGE nell'Appendice A) o anche di leggere il programma esattamente come ogni altro file di dati in formato ASCII.

## COME CARICARE I PROGRAMMI

Se volete caricare da disco un file contenente un programma BASIC, inserite il disco in un drive e, quando il disco è pronto, inviate il comando LOAD, con la sintassi:

LOAD *"specfile"* [,R]

*specfile* dev'essere un nome valido del tipo di dispositivo che state utilizzando: se viene tralasciata l'indicazione del dispositivo si assume per default il drive corrente, di solito il drive A:.
L'opzione R del comando LOAD fa sì che il programma indicato venga mandato immediatamente in esecuzione dopo essere stato caricato nella memoria del PC. Il comando:

```
LOAD "B:PROGRAM.BAS" ,R
```

perciò, è identico a:

```
LOAD "B:PROGRAM.BAS"
Ok
RUN
```

## COME ESEGUIRE I PROGRAMMI

Una volta che un programma BASIC si trova in memoria, potete avviarne l'esecuzione con il comando RUN che ha la sintassi:

RUN [*linea*]

oppure:

RUN *"specfile"*

Se viene inviato il comando senza parametri, viene eseguito il programma in memoria a partire dalla linea con il numero più basso; se viene indicato *linea*, da lì parte l'esecuzione.

Se si indica *specfile* il BASIC carica automaticamente il file dal dispositivo nominato o dal drive corrente ed esegue il programma a partire dalla linea con il numero più basso.
Se non vengono indicate estensioni nel nome del file, viene assunta per default l'estensione .BAS.

Capitolo

# Fondamenti di BASIC

# 5

Nel capitolo precedente abbiamo visto come creare programmi BASIC con il PC: in questo introdurremo invece le parole, i simboli e le regole del linguaggio BASIC.

## 5.1 I dati

Un programma in BASIC serve per manipolare informazioni, che prendono il nome di dati; questi possono essere composti sia da numeri che da caratteri.

### DATI COMPOSTI DA CARATTERI: LE STRINGHE

Una *stringa* è, in BASIC, una sequenza di caratteri che inizia e termina con le virgolette. Ecco alcuni validi esempi di stringhe:

"Non è chiaro il procedimento della spiegazione"
"$68.45"
"RELAZIONE CONCLUSIVA"

Le stringhe possono raggiungere un massimo di 255 caratteri, scelti tra tutti quelli che trovate nell'Appendice B tranne, naturalmente, le virgolette. Notate come alcuni di questi caratteri non appaiano sulla tastiera: per inserirli in una stringa di caratteri, dovete tenere premuto il tasto ALT

mentre battete il codice del carattere in questione, che trovate nell'Appendice B. Se vi interessasse, ad esempio, la lettera greca sigma (Σ) dovreste battere il numero 228 sul tastierino numerico mentre tenete premuto il tasto ALT.

## DATI NUMERICI: I NUMERI

Il BASIC riconosce queste diverse categorie di numeri:

— Interi
— Reali in virgola fissa
— Reali in virgola mobile
— Numeri esadecimali
— Numeri ottali

### Interi

Gli interi BASIC sono tutti i numeri interi compresi tra −32768 e +32767; i numeri negativi devono sempre essere preceduti dal segno meno (−), mentre il segno più (+) è facoltativo, in quanto un numero non preceduto da segno viene sempre considerato positivo. Non sono permesse virgole in nessun tipo di numeri. Ecco alcuni esempi, validi e non, di interi:

| | |
|---|---|
| 4 | valido |
| −30000 | valido |
| 1777 | valido |
| 12,840 | non valido: non è permessa la virgola |
| 32840 | non valido: è troppo grande |
| −99999 | non valido: è troppo piccolo |

### Reali in virgola fissa

I numeri in virgola fissa sono numeri reali, che possono variare tra ±99999999999999999 (diciassette 9). Un'osservazione: il BASIC può operare internamente con numeri lunghi fino a 17 cifre, ma non ne verranno mai visualizzate più di 16; vedremo più tardi le conseguenze di questo fatto.
I reali in virgola fissa possono avere cifre sia alla destra che alla sinistra del punto decimale e sono composti al massimo da diciassette cifre. Ecco alcuni esempi, validi e non, di reali in virgola fissa:

| | |
|---|---|
| 345.234234 | valido |
| −94949494949.494 | valido |
| 0.000000000000001 | valido |
| 0.0000000000000001 | non valido: troppe cifre |
| 4,509 | non valido: non è permessa la virgola |

### Reali in virgola mobile

I numeri in virgola mobile sono reali espressi in forma esponenziale (detta anche notazione scientifica); in questa forma ogni numero è composto da due parti: la mantissa e l'esponente, preceduto dalla lettera E o dalla D. Il valore di un numero in virgola mobile si calcola moltiplicando la mantissa per dieci elevato all'esponente indicato, come vedete negli esempi seguenti:

```
345.44E3=(345.44)*10^3
        =(345.44)*10*10*10
        =345440

-6.92341E-3=(-6.92341)*10^-3
           =(-6.92341)*0.001
           =-0.00692341
```

I numeri in virgola mobile in BASIC possono variare tra 10E−38 e 10E+38 oppure tra −10E−38 e −10E38.
In generale i numeri in virgola mobile vengono usati per rappresentare valori molto piccoli o molto grandi, soprattutto quando sono composti da un numero di cifre troppo alto per essere rappresentato in virgola fissa.

### Numeri esadecimali

Il BASIC riconosce i numeri esadecimali, cioè numeri scritti in base 16, lunghi fino a quattro cifre. Per comporre i numeri esadecimali si usano le cifre da 0 a 9 e le lettere A, B, C, D, E ed F. I numeri esadecimali devono essere sempre preceduti dal simbolo &H. Potete vedere ora alcuni esempi di numeri esadecimali validi.

| | |
|---|---|
| &H1 | (=1 in decimale) |
| &HA92 | (=2706 in decimale) |
| &HC017 | (=49175 in decimale) |

I numeri esadecimali vengono solitamente usati per interagire con l'hardware del PC dall'interno di un programma BASIC.

### Numeri ottali

Il BASIC riconosce anche i numeri ottali (in base 8), che devono essere preceduti dal simbolo & e possono avere al massimo sei cifre, comprese tra 0 e 7. Alcuni esempi di numeri ottali validi in BASIC sono:

| | |
|---|---|
| &340 | (=224 in decimale) |
| &7 | (=7 in decimale) |
| &111121 | (=37457 in decimale) |

## PRECISIONE DEI NUMERI IN BASIC

Per precisione di un numero, in BASIC, si intende il numero di cifre utilizzato per rappresentarlo. Se diciamo, infatti, che un numero ha sei cifre significative, vogliamo intendere che solo le prime sei cifre del numero sono accurate; ogni altra cifra è usata solo per l'arrotondamento.
In BASIC sono possibili tre livelli di precisione: precisione *intera*, usata per interi, esadecimali ed ottali, precisione *semplice* e *doppia* precisione entrambe usate per numeri reali in virgola fissa e mobile.

### Precisione intera

Se un numero è intero, esadecimale o ottale, tutte le sue cifre sono accurate, sempre che, naturalmente, il numero appartenga all'intervallo di variazione precedentemente definito per questi tipi di numeri. Se invece il risultato di un calcolo in cui compaiono numeri con precisione intera non è un intero, il risultato finale sarà accurato entro ±0.5 se il valore viene arrotondato ad un intero ed entro ±1 se viene troncato ad un intero. La differenza tra il troncamento e l'arrotondamento è illustrata qui di seguito:

| **Numero originale** | **Arrotondamento** | **Troncamento** |
|---|---|---|
| 3.00 | 3 | 3 |
| 3.01 | 3 | 3 |
| 3.49 | 3 | 3 |
| 3.50 | 4 | 3 |
| 3.99 | 4 | 3 |
| −3.50 | −4 | −3 |
| −3.49 | −3 | −3 |

## Precisione semplice

I numeri in precisione semplice vengono rappresentati in BASIC con sette cifre, di cui le prime sei sono accurate. Un numero viene considerato in precisione semplice se, innanzitutto, non è un intero, un esadecimale e un ottale e se è vera almeno una delle seguenti affermazioni:

— Il numero è composto da sette cifre o meno
— Il numero è seguito dal segno !
— Il numero è in virgola mobile e prima dell'esponente appare la lettera E

I seguenti sono esempi di numeri in precisione semplice:

847.99
9!
−1234.34E4

Per comprendere appieno il significato della precisione in un calcolo con precisione semplice osservate l'esempio seguente:

```
Ok
PRINT 234.44/3
78.1466
Ok
```

A causa della precisione semplice sono accurate solo le prime sei cifre del risultato, ma potete sfruttare la settima cifra per arrotondare il numero a 78.1467. La risposta corretta sarebbe 78.14666... (con una sequenza infinita di sei): se arrotondate questo numero alla lunghezza di sei cifre, ottenete 78.1467, in accordo con il risultato appena trovato.

## Doppia precisione

I numeri in doppia precisione vengono memorizzati dal PC con 17 cifre, di cui solo 16 vengono visualizzate (o in genere utilizzate per l'output) e tutte e sedici sono accurate.
Perché un numero sia considerato in doppia precisione deve valere almeno una delle seguenti affermazioni:

— Il numero è composto da otto o più cifre
— Il numero è seguito dal simbolo #
— Il numero è in virgola mobile e la lettera che precede l'esponente è una D

Ecco alcuni esempi di numeri in doppia precisione:

4#
12345678
12345678901234567
−234.90009D3

## 5.2 Le variabili

Sono due i modi in cui i dati possono essere usati all'interno dei programmi BASIC: o vengono esplicitamente definiti come costanti, come segue

```
PRINT 82746.7888
82746.7888
Ok
```

oppure ci si riferisce ad essi con un nome, detto *variabile*:

```
X#=82746.7888
Ok
PRINT X#
82746.7888
Ok
```

In questo caso alla variabile X# è stato assegnato il valore, in doppia precisione, 82746.7888.
Le variabili servono per rappresentare sia dati numerici che dati di tipo carattere (stringhe). Provate ad eseguire questo breve programma:

```
10 NUMERO1=3
20 NUMERO2=5
30 COMPUTER$="PC XT dell'IBM"
40 PRINT "Il ",COMPUTER$; "puo' visualizzare var
iabili di tipo stringa."
50 PRINT NUMERO1*NUMERO2
60 END
```

che usa due variabili numeriche, NUMERO1 e NUMERO2, ed una di tipo stringa, COMPUTER$.
Le variabili sono molto utili perché vi permettono di sostituire i valori effettivi o le stringhe con "etichette" (i nomi delle variabili) che alludono al

significato reale del numero o della stringa; permettono inoltre di utilizzare lo stesso programma con un grande numero di dati diversi. Per esempio:

```
2000 PRINT "Il numero di miglia equivalente e' "
     4294*.6 " miglia."
```

Qui il valore della distanza espresso in chilometri è usato come costante; la conversione in miglia (cioè la moltiplicazione del numero di chilometri per 0.6) viene eseguita all'interno dell'istruzione PRINT. Un modo decisamente migliore per eseguire questa operazione è quello che usa le variabili, come potete vedere qui di seguito:

```
800 CHILOM=4294
810 UNIT$="miglia"
820 MIGLIA=CHILOM*.6
      .
      .
      .
830 PRINT "Il numero di "UNIT$;"equivalente e' "
MIGLIA UNIT$
```

Abbiamo sostituito delle variabili a tre parti dell'istruzione PRINT dell'esempio precedente: innanzitutto l'espressione matematica 4294*.6 è stata sostituita dalla variabile MIGLIA, che dà al lettore del programma un'idea chiara di cosa significhi quel valore; in secondo luogo abbiamo sostituito alla costante 4294 la variabile CHILOM; infine abbiamo sostituito la parola "miglia" con la variabile UNIT$.
A questo punto possiamo usare la stessa istruzione PRINT per visualizzare i risultati di diverse conversioni cambiando semplicemente il valore di CHILOM e di UNIT$.

## NOMI DI VARIABILI

Dovete rispettare alcune convenzioni nell'assegnare nomi alle variabili.
In primo luogo un nome può essere lungo al più 40 caratteri e deve sempre iniziare con una lettera dell'alfabeto; per il resto del nome sono permesse solo lettere, numeri ed il punto, tranne che per l'ultimo carattere, che può essere un simbolo speciale per indicare la precisione della variabile, come vedremo.
Inoltre, un nome non può essere una delle parole chiave del BASIC (dette anche "parole riservate"), né una parola chiave seguita dai simboli $, %, !, #. Tra le parole chiave sono compresi comandi, istruzioni, nomi di fun-

zioni e nomi di operatori. I nomi possono, però, contenere parole chiave; vediamo perciò che

FOR
FOR!

sono nomi non validi, mentre

FORMAGGIO!

è perfettamente lecito.
Come terzo punto, un nome di variabile che inizi con le lettere FN è valido solo nel caso sia stato definito come "funzione definita dall'utente". In questo stesso capitolo, più avanti, tratteremo più esplicitamente questo tipo di funzioni.
Ultimo, ma non meno importante, il nome stesso serve per definire il tipo e la precisione della variabile: questa azione è compiuta da quello che viene detto *carattere di dichiarazione del tipo di variabile* ($, %, !, #) al termine del nome stesso.

**Nomi di variabili di tipo stringa**

Un nome di variabile seguito dal simbolo $ definisce una variabile di tipo stringa; eccone alcuni esempi:

E$
INDIRIZZO$
ASSOCIAZIONIPOLITICHE$
ABRACADABRA13$

Esiste un diverso modo per definire variabili di tipo stringa, per mezzo dell'istruzione DEFSTR (vedi Appendice A).

**Nomi di variabili intere**

Il simbolo % al termine di un nome di variabile indica una variabile che assume valori interi:

ANNO%
SPIN%
NOMEDIVARIABILEINTERARIDICOLOMAVALIDO%

Un altro modo per definire variabili intere è di usare l'istruzione DEFINT, come potete vedere nell'Appendice A.

### Nomi di variabili in precisione semplice

I nomi di variabili in precisione semplice terminano col simbolo ! o con nessun simbolo particolare; tutti i seguenti sono nomi validi per queste variabili:

ALTEZZAMETRI!
AMMONTAREDEBITO
NUMERODIGIORNIDOPOIL9.9.57!

L'istruzione DEFSNG può essere utilizzata in alternativa per definire variabili in precisione semplice (vedi Appendice A).

### Nomi di variabili in doppia precisione

L'ultimo carattere di un nome di variabile in doppia precisione dev'essere il simbolo #:

PI#
VELOCITASUONO#
FREQUENZA#

Per ottenere variabili in doppia precisione potete usare anche l'istruzione DEFDBL (Appendice A).

### Come cambiare la precisione

Quando assegnate un valore di una determinata precisione ad una variabile di precisione più bassa, il numero viene arrotondato in modo da raggiungere la precisione implicata dal nome di variabile; l'istruzione

```
X=45.9039852
```

assegna il valore 45.90399 alla variabile in precisione semplice X.
Osservate invece questa istruzione:

```
T%=32.76724E3
```

che assegna alla variabile intera il valore 32767.

Infine considerate l'istruzione seguente:

```
S%=32.76755E3
```

che ha come risultato un messaggio di errore di overflow (superamento della massima capacità) poiché il numero arrotondato, 32768, è troppo grande per essere rappresentato da una variabile intera.
Nelle operazioni in cui combinate costanti e variabili di diversa precisione, tutte queste vengono trattate come se avessero la massima precisione rappresentata; anche il risultato viene espresso con questa precisione:

```
Ok
A%=3
Ok
PRINT A%/4.44#
.6756756756756757
Ok
```

In questo caso la precisione più alta che compare nell'istruzione PRINT è data dalla costante in doppia precisione 4.44#; questo ha l'effetto che il BASIC considera la variabile intera A% come se fosse anch'essa in doppia precisione. Il risultato, 0.6756756756756757, è visualizzato come numero in doppia precisione, sempre ricordando che le cifre che compaiono sono 16, mentre quelle memorizzate sono 17.

## 5.3 Gli array

Viene definito *array* un insieme di variabili, tutte dello stesso tipo, che abbiano un nome comune. Supponete di avere un programma che elabori liste di nominativi di studenti; potreste identificare questi nominativi con un insieme di nomi di variabile di tipo stringa, ciascuno contenente un solo nominativo, come segue:

```
10 STUDENTEUNO$ = "Alberto Alboini"
20 STUDENTEDUE$ = "Arrigo Arverni"
                .
                .
                .
200 STUDENTEVENTI$ = "Zaccaria Zuccoli"
```

Un modo migliore per risolvere il problema è quello di usare un array per tutti i nominativi degli studenti: disponendo di un array voi vi riferi-

te ad una singola variabile (detta *elemento* dell'array) specificando il nome dell'array stesso, seguito da un *indice* che corrisponde alla posizione nell'array della variabile che interessa. Nell'esempio precedente possiamo definire l'array STUDENTE$(*n*), con *n* indice dell'array:

```
5 DIM STUDENTE$(20)
10 STUDENTE$(0) = "Alberto Alboini"
20 STUDENTE$(1) = "Arrigo Arverni"
                .
                .
                .
200 STUDENTE$(19) = "Zaccaria Zuccoli"
```

L'istruzione DIM è stata posta qui semplicemente per riservare in memoria spazio sufficiente a contenere i venti elementi dell'array; osservate, a questo proposito, che la numerazione degli elementi parte da 0: questa è la condizione normale, che può essere modificata in maniera tale che il più basso elemento dell'array sia il numero 1 (vedi l'istruzione OPTION BASE nell'Appendice A).
Gli array permettono anche di recuperare e riutilizzare dati nei programmi: a titolo di esempio consideriamo la routine che visualizza la lista dei nominativi degli studenti.

```
580 FOR X=0 TO 19
590 PRINT STUDENTE$(X)
600 NEXT
```

Le linee che contengono i comandi FOR X=0 TO 19 e NEXT fanno sì che la linea 590 sia eseguita in totale 20 volte, in ciascuna delle quali il valore viene incrementato di uno, da 0 a 19.

## ARRAY CON PIÙ DI UNA DIMENSIONE

Il nome di un array può essere seguito da più di un indice, ciascuno dei quali rappresenta una *dimensione* dell'array stesso. Il numero di dimensioni di un array può variare da 1 a 255, mentre il numero di elementi che si riferiscono alla stessa dimensione può raggiungere al massimo 32767. L'array STUDENTE$(*n*) dell'esempio precedente aveva una sola dimensione, poiché doveva rappresentare una lista di nomi.
Gli array in due dimensioni permettono di strutturare i dati in matrice, cosa che potrebbe servire per memorizzare l'indirizzo di ogni studente a fianco del nome: creiamo a questo scopo l'array STUDENTE$(*n,m*), che appare come segue:

| | |
|---|---|
| STUDENTE$(0,0) | STUDENTE$(0,1) |
| STUDENTE$(1,0) | STUDENTE$(1,1) |
| ⋮ | ⋮ |
| STUDENTE$(*n*,0) | STUDENTE$(*n*,1) |

in cui i nomi degli studenti appaiono nella prima colonna, mentre la seconda contiene i rispettivi indirizzi. Voi utilizzerete l'indice *n* per identificare lo studente e l'indice *m* per scegliere nome o indirizzo: in questo modo, STUDENTE$(1,1) fornisce l'indirizzo del secondo studente.
Come esempio di array tridimensionale potete pensare all'array LIBRO(*n*,*r*,*p*), usato per memorizzare brani di testo: l'indice *n* indica il numero della pagina, l'indice *r* una riga all'interno della pagina e l'indice *p* una singola parola all'interno della riga.

## 5.4 Le espressioni

Si definisce *espressione* BASIC una singola variabile o costante, oppure ogni combinazione di variabili, costanti, operatori e funzioni; ecco alcuni esempi di queste espressioni:

```
NOTA%
Y+43.2292
SPESA*(TASSO/100)
```

Ogni valore che compare in un'espressione viene detto *operando*, mentre i simboli che definiscono l'operare dell'espressione sono detti *operatori*.

### GLI OPERATORI

Gli operatori indicano le operazioni applicate sui dati di un'espressione: l'operatore *, ad esempio, indica la moltiplicazione.

### OPERATORI ARITMETICI

Tra gli operatori un posto di rilievo è occupato dagli operatori aritmetici, elencati nella Tabella 5.1 insieme al rispettivo modo di operare.

**Tabella 5.1** Gli operatori aritmetici

| Operatore | Operazione |
|---|---|
| ^ | elevazione a potenza |
| − | negazione |
| * | moltiplicazione |
| / | divisione in virgola mobile |
| \ | divisione intera |
| MOD | modulo aritmetico |
| + | addizione |
| − | sottrazione |

Nella tabella gli operatori sono elencati nell'ordine di priorità, cioè l'ordine in cui le operazioni vengono eseguite se l'espressione contiene più di un operatore. Tenete conto, però, che l'operatore di moltiplicazione e quello di divisione in virgola mobile hanno lo stesso ordine di priorità, così come gli operatori di addizione e sottrazione, come accade nell'algebra.

## Elevazione a potenza

Alcuni esempi di espressioni contenenti l'operatore di elevazione a potenza possono essere i seguenti:

```
234992.234444^.837
LATOCUBO^3
5.234E3^EXPO#
```

Nell'ultima espressione la costante 5.234E3 è elevata al valore della variabile EXPO#. Supponendo che questo sia uguale a 4.2875, l'espressione verrà calcolata in questo modo:

```
5.234E3^EXPO# = 5.234E3^4.2875
              = (5.234*1000)^4.2875
              = 5234^4.2875
              = 8800314000000000
              = 8.800314E15
```

Questo è il valore utilizzato dal BASIC, a meno che il risultato della espressione non venga assegnato ad una variabile intera o in precisione semplice

### Negazione

Con il termine negazione si indica l'operazione di cambiamento del segno; ogni numero preceduto dal segno meno sarà considerato con il valore negativo:

| | | |
|---|---|---|
| −(−395) | = 395 | |
| −ENTRATE | = −20.89 | Se il valore iniziale di ENTRATE era 20.89 |
| −ENTRATE | = 386.29 | Se il valore iniziale di ENTRATE era −386.29 |

### Moltiplicazione

La moltiplicazione come già abbiamo visto, è rappresentata dall'operatore asterisco (*):

```
OGGETTO%(X)*PESOA!(X)
QUANTITA*PREZZOUNIT
```

### Divisione in virgola mobile

È rappresentata dal simbolo slash (/): il risultato (quoziente) di una divisione in virgola mobile è un numero reale, rappresentato in virgola mobile se la sua grandezza lo consente e con una precisione uguale alla massima precisione del dividendo o del divisore. Ad esempio:

```
3#/DIVISORE%
```

darà un quoziente in doppia precisione, mentre

```
83/PIPPO!
```

darà un quoziente in precisione semplice.

### Divisione intera

Per eseguire la divisione intera, rappresentata dall'operatore backslash ( \ ), il BASIC dapprima converte il divisore ed il dividendo in interi, arrotondandoli se necessario e poi esegue la divisione: il quoziente viene troncato ad un intero. Potete seguire il meccanismo or ora descritto nell'espressione seguente:

269.9 \ 3 = 270 \ 3
= 90

268.9 \ 3 = 269 \ 3
= 89.66666666...
= 89

Vi presentiamo ora un problema che spiega la necessità di introdurre un operatore come la divisione intera; supponiamo di voler distribuire un numero X di mele in un numero Y di cestini, in modo che tutti i cestini contengano lo stesso numero di mele; poiché non potete suddividere una singola mela in parti più piccole, il quoziente di una divisione in virgola mobile dovrebbe essere successivamente arrotondato ad un intero. La divisione intera esegue automaticamente questa azione.

### Modulo aritmetico

Potete usare l'operatore MOD per ottenere il resto (intero) di una divisione intera.
L'operazione modulo aritmetico viene esemplificata nelle righe seguenti:

82 MOD 9 = resto della divisione (82 \ 9)
= 1

6.43 MOD 8 = 6

Nel secondo esempio il risultato 6, rappresenta il resto, troncato, di 6 diviso 8.
Osservate che l'operatore MOD deve essere separato da almeno uno spazio dal secondo operando, altrimenti il BASIC lo considera una variabile.

### Addizione e sottrazione

L'addizione e la sottrazione sono rappresentate, rispettivamente, dagli operatori + e −. Quando viene usato come operatore di negazione, il simbolo − è equivalente ad una sottrazione dell'operando dal numero 0 o, in altri termini:

−VARIABILE = 0−VARIABILE

## OPERATORI RELAZIONALI

Un tipo particolare di operatori sono gli operatori relazionali, che agiscono sia su numeri che su stringhe per confrontare due operandi dello stes-

so tipo. Il loro risultato può essere "vero" o "falso", valori logici rappresentati, in BASIC, da $-1$ (vero) e 0 (falso). Gli operatori e la loro azione sono elencati nella Tabella 5.2.

**Tabella 5.2.** Gli operatori relazionali

| Operatore | Condizione |
|---|---|
| = | Uguale |
| <> o >< | Diverso |
| < | Minore di |
| > | Maggiore di |
| <= o =< | Minore o uguale |
| >= o => | Maggiore o uguale |

Questi operatori non hanno alcun particolare ordine di precedenza: vengono valutati da sinistra a destra come compaiono nell'espressione (sempre che non siano presenti delle parentesi nell'espressione, come vedremo). Gli operatori relazionali vengono solitamente usati per controllare alcune condizioni. Per esempio:

```
        .
        .
        .
230 IF RISPOSTA>99 THEN 1000
240 PRINT "La RISPOSTA va bene."
        .
        .
        .
1000 RISPOSTA=0
1010 PRINT "La RISPOSTA era troppo grande ed e'
stata posta uguale a 0."
        .
        .
```

L'espressione in linea 230 controlla il valore della variabile RISPOSTA: se questo è minore o uguale a 99 viene eseguita la linea 240, altrimenti l'esecuzione salta alla linea 1000.

### Come combinare operatori aritmetici e relazionali

I due tipi di operatori finora incontrati, operatori aritmetici e relazionali, possono essere combinati in espressioni più complesse:

VALOREDELTEST> =(69*ALTEZZA)

confronta il risultato dell'espressione aritmetica 69*ALTEZZA con il valore corrente della variabile VALOREDELTEST.
Ecco un altro esempio:

CLASSIFICA+(PUNTEGGIO>450)+(HANDICAP< >3)

Questa espressione ha come risultato la somma del valore di CLASSIFICA con i risultati delle espressioni relazionali PUNTEGGIO>450 e HANDICAP< >3. In altre parole, se assumiamo come valore di CLASSIFICA 38, il valore dell'espressione dipenderà dai valori di PUNTEGGIO e HANDICAP come segue:

| **Valore degli operandi** | **Valore dell'espressione** |
|---|---|
| PUNTEGGIO < =450, HANDICAP = 3 | 38+ 0+ 0 = 38 |
| PUNTEGGIO < =450, HANDICAP < >3 | 38+ 0+ −1 = 37 |
| PUNTEGGIO > 450, HANDICAP = 3 | 38+ −1+ 0 = 37 |
| PUNTEGGIO > 450, HANDICAP < >3 | 38+ −1+ −1 = 36 |

Notate che le espressioni tra parentesi in questo esempio vengono calcolate per prime: questo accade perché l'ordine di precedenza degli operatori è sempre subordinato alla presenza delle parentesi nell'espressione.

### Operatori relazionali che agiscono su stringhe

Quando un operatore relazionale viene fatto agire su due stringhe, ogni carattere della prima stringa viene confrontato con il corrispondente carattere della seconda, a partire dall'inizio della stringa stessa: non appena si trova una discrepanza tra le due stringhe, il BASIC dà il risultato, basandosi sui due caratteri coinvolti; se una delle due stringhe termina prima dell'altra, ma senza che siano state riscontrate differenze, viene considerata minore la stringa più corta.
L'ordine di grandezza assegnato alle lettere e ai numeri presenti in una stringa è il seguente:

0,1,2,3,...,8,9,A,B,C,..., X,Y,Z,a,b,c,...,x,y,z
minore ——————————→ maggiore

Infatti il BASIC per confrontare due caratteri altro non fa che confrontare i codici ASCII dei caratteri stessi: se osservate l'Appendice B, vi accorgete subito di qual è l'ordine dei numeri e delle lettere basato sul loro codice ASCII.

Ecco un esempio per illustrare la manipolazione del BASIC di questi operatori relazionali sulle stringhe:

| **Espressione** | **Risultato** |
|---|---|
| "BASTO">"BASTONE" | 0 |
| "$TAX"<"%TAX" | −1 |
| "Gli spazi contano">= "Gli spazi contano " | 0 |

## OPERATORI LOGICI

Gli operatori logici eseguono operazioni di algebra booleana sui loro operandi; l'algebra booleana definisce un insieme di relazioni tra due operandi, in cui questi possono essere valutati come veri o falsi: anche qui, perciò, il risultato dell'operazione sarà vero o falso. Questi operatori vengono usualmente impiegati nei programmi quando si tratta di prendere delle decisioni.
Gli operatori logici del BASIC sono elencati in Tabella 5.3 nel loro ordine di precedenza, mentre la loro azione su un'espressione è mostrata nella "Tabella della verità" riportata in Figura 5.1.

**Tabella 5.3** Gli operatori logici

| Operatore | Funzione logica |
|---|---|
| NOT | Complemento logico |
| AND | Congiunzione |
| OR | Disgiunzione |
| XOR | OR esclusivo |
| IMP | Implicazione |
| EQV | Equivalenza |

Il risultato di un operatore logico in BASIC è dato da −1 (vero) o da 0 (falso); ad esempio:

INDICE% > 25 AND LIMITE = 1.789

sarà sostituita dal valore −1 se INDICE% è maggiore di 25 e LIMITE è uguale a 1.789, altrimenti il BASIC valuterà l'espressione 0, poiché sarebbe falsa. Osservate come la stessa espressione contiene sia operatori relazionali che operatori logici: quando viene valutata, vengono calcolate dapprima le due espressioni relazionali ed i loro risultati vengono utilizzati come operandi per l'operatore logico AND.

| Operatore | Tabella della verità | | |
|---|---|---|---|
| NOT | **X** | | **NOT X** |
| | V | | F |
| | F | | V |
| AND | **X** | **Y** | **X AND Y** |
| | V | V | V |
| | V | F | F |
| | F | V | F |
| | F | F | F |
| OR | **X** | **Y** | **X OR Y** |
| | V | V | V |
| | V | F | V |
| | F | V | V |
| | F | F | F |
| XOR | **X** | **Y** | **X XOR Y** |
| | V | V | F |
| | V | F | V |
| | F | V | V |
| | F | F | F |
| IMP | **X** | **Y** | **X IMP Y** |
| | V | V | V |
| | V | F | F |
| | F | V | V |
| | F | F | V |
| EQV | **X** | **Y** | **X EQV Y** |
| | V | V | V |
| | V | F | F |
| | F | V | F |
| | F | F | V |

**Figura 5.1** La tabella della verità per gli operatori logici (V=vero, F=falso)

## Operatori logici che agiscono su numeri

Gli operandi degli operatori logici possono anche essere dei numeri: il BASIC li converte in interi e controlla che il loro valore sia compreso nell'intervallo permesso per i numeri interi in BASIC (tra −32768 e +32767).

L'operazione logica viene poi eseguita utilizzando la rappresentazione binaria (o in base 2) degli operandi; vi mostriamo qui alcuni numeri decimali nella loro forma binaria:

| Numero decimale | Rappresentazione binaria |
|---|---|
| 0 | 0 |
| 1 | 1 |
| 2 | 10 |
| 31 | 11111 |

Un numero binario è composto da una sequenza di 1 e di 0, in pratica da una serie di valori veri e falsi.
Gli operatori logici trattano la sequenza di 1 e 0 (o sequenza di bit) di un numero binario in modo del tutto simile a quello usato dagli operatori relazionali per agire sulle stringhe; un'operazione logica, cioè, viene eseguita su ogni coppia di bit che occupi la stessa posizione relativa all'interno delle due sequenze (dei due operandi), ed il risultato di ogni singola operazione è utilizzato per formare il risultato finale:

```
250 AND 28 = 11111010 AND 00011100
           = 00011000
           = 24
```

**CONCATENAZIONE**

L'operatore +, che abbiamo già incontrato come operatore di addizione, viene usato anche come operatore di concatenazione quando gli operandi sono delle stringhe; concatenazione significa che le due stringhe che si trovano ai lati del segno + vengono unite a formare un'unica stringa più lunga; nella stessa operazione di concatenazione possono essere inserite anche più di due stringhe:

```
10 BOILERPLATE1$= "Mentre "
20 BOILERPLATE2$= " ricerca solo la verita' e la
giustizia......... "
30 PRINT BOILERPLATE1$+"Herbert P. Jones"+BOILER
PLATE2$

"Mentre Herbert P. Jones ricerca solo la verita'
e la giustizia.........
```

## 5.5 Le funzioni

Le *funzioni* BASIC vengono usate per eseguire calcoli numerici o su stringhe, oppure per controllare l'hardware del PC od ottenerne informazioni. In generale le funzioni operano su di un argomento, cioè su espres-

sioni specificate insieme alla funzione; alcune funzioni, però, non richiedono argomenti.
Possiamo dividere in tre categorie le funzioni usate nei calcoli: le funzioni numeriche, le funzioni che operano sulle stringhe e le funzioni "definite dall'utente".

## FUNZIONI NUMERICHE

Le funzioni numeriche hanno, come dice il nome, un risultato numerico. Il BASIC dispone di un grande numero di queste funzioni predefinite, che potete trovare elencate nella Tabella 5.4. (Vedi Appendice A per ulteriori dettagli su queste funzioni).

### Funzioni numeriche matematiche

Molte delle funzioni numeriche del BASIC sono le ordinarie funzioni matematiche, come ad esempio:

| | |
|---|---|
| `PRINT SIN(ANGOLO)` | Visualizza il seno del valore della variabile ANGOLO che deve essere espressa in radianti |
| `PRINT ABS(TEST#)` | Visualizza il valore assoluto del valore della variabile TEST# |

Alcune funzioni sono utilizzate per convertire valori BASIC da una precisione ad un'altra. Ad esempio:

| | |
|---|---|
| `PRINT CDBL(INTERO%)` | Visualizza il valore di INTERO% in doppia precisione |

### Funzioni numeriche riferite a stringhe

Un certo numero di funzioni numeriche BASIC agiscono sulle stringhe ma restituiscono un risultato numerico:

| | |
|---|---|
| `PRINT LEN(PRIMARIGA$)` | Visualizza la lunghezza (il numero di caratteri) della stringa PRIMARIGA$ |

**Tabella 5.4** Funzioni numeriche

| **Funzioni aritmetiche** | **Risultato** |
|---|---|
| ABS(*x*) | Valore assoluto di *x* |
| ATN(*x*) | Arcotangente (in radianti) di *x* |
| CDBL(*x*) | Converte *x* in un numero in doppia precisione |
| CINT(*x*) | Converte *x* in un intero per arrotondamento |
| COS(*x*) | Coseno dell'angolo *x*, con *x* in radianti |
| CSNG(*x*) | Converte *x* in un numero in precisione semplice |
| EXP(*x*) | Eleva e alla potenza *x* |
| FIX(*x*) | Tronca *x* ad intero |
| INT(*x*) | L'intero più grande minore o uguale a *x* |
| LOG(*x*) | Logaritmo naturale di *x* |
| RND(*x*) | Numero casuale |
| SGN(*x*) | Segno di *x* |
| SIN(*x*) | Seno dell'angolo *x*, con *x* in radianti |
| SQR(*x*) | Radice quadrata di *x* |
| TAN(*x*) | Tangente dell'angolo *x*, con *x* in radianti |
| **Funzioni riferite a stringhe** | |
| ASC(*x*$) | Codice ASCII del primo carattere nella stringa *x*$ |
| CVI(*x*$) CVS(*x*$) CVD(*x*$) | Convertono *x*$ in un numero in precisione intera, semplice, doppia |
| INSTR(*n*,*x*$,*y*$) | Posizione della prima occorrenza di *y*$ nella stringa *x*$ a partire dalla posizione *n* |
| LEN(*x*$) | Lunghezza della stringa *x*$ |
| VAL(*x*$) | Valore numerico della stringa *x*$ |
| **Funzioni I/O e varie** | |
| CSRLIN | Restituisce la posizione verticale del cursore |
| EOF(*f*) | Indica una condizione end-of-file nel file *f* |
| ERL | Restituisce il numero di linea a cui è stato trovato l'ultimo errore (vedi ERR) |
| ERR | Restituisce il codice d'errore dell'ultimo errore |
| FRE(*x*$) | Restituisce la quantità di spazio libero in memoria non utilizzato dal BASIC nella corrente sessione |
| INP(*n*) | Legge un byte dalla porta *n* |
| LOC(*f*) | Restituisce la locazione del file *f*: — numero del successivo record in file ad accesso diretto |

**Tabella 5.4** Funzioni numeriche (continua)

| **Funzioni I/O e varie** | **Risultato** |
|---|---|
| | — numero di settori letti o scritti per file sequenziali<br>— numero di caratteri nel buffer di comunicazione in input |
| LOF(*f*) | Restituisce la lunghezza del file *f*:<br>— numero di byte (in multipli di 128) in file sequenziali o ad accesso diretto<br>— numero di byte liberi nel buffer di comunicazione in input |
| LPOS(*n*) | Restituisce la posizione del carrello della stampante |
| PEEK(*n*) | Legge il byte nella locazione di memoria *n* |
| PEN(*n*) | Legge la penna ottica |
| PLAY(*n*) | Numero di note nel buffer Accompagnamento musicale |
| PMAP(*x,n*) | Trasforma le coordinate dal sistema WINDOW al sistema VIEW |
| POINT(*n*) | Valore della coordinata grafica corrente (*x* o *y*) (LRP) |
| POINT(*x,y*) | Restituisce il colore del punto (*x,y*) (in modo Grafico) |
| POS(*n*) | Restituisce la posizione orizzontale del cursore |
| SCREEN(*ri,col,z*) | Restituisce il carattere o il colore nella posizione (*ri,col*) |
| STICK(*n*) | Restituisce le coordinate di un joystick |
| STRIG(*n*) | Restituisce lo stato del pulsante del joystick |
| TIMER(*n*) | Numero di secondi trascorsi dalla mezzanotte o dall'ultima reinizializzazione del sistema |
| USR*n*(*x*) | Chiama una routine in linguaggio macchina con argomento *x* |
| VARPTR(*var*) | Restituisce l'indirizzo in memoria della variabile *var* |
| VARPTR(#*f*) | Restituisce l'indirizzo del blocco di controllo per il file *f* |

## FUNZIONI DI TIPO STRINGA

Il BASIC dispone, inoltre, di un certo numero di funzioni che hanno come risultato una stringa: queste vengono elencate nella Tabella 5.5.

**Tabella 5.5** Funzioni di tipo stringa

| Funzioni | Risultato |
|---|---|
| CHR$(*n*) | Carattere con codice ASCII *n* |
| LEFT$(*x*$,*n*) | Gli *n* caratteri più a sinistra della stringa *x*$ |
| MID$(*x*$,*n*,*m*) | *m* caratteri della stringa *x*$ a partire dalla posizione *n* |
| RIGHT$(*x*$,*n*) | Gli *n* caratteri più a destra della stringa *x*$ |
| SPACE$(*n*) | *n* spazi bianchi |
| STRING$(*n*,*x*$) | Primo carattere di *x*$ ripetuto *n* volte |
| STRING$(*n*,*m*) | Carattere con codice ASCII *m*, ripetuto *n* volte |
| **Funzioni I/O e varie** | |
| DATE$ | Restituisce la data di sistema |
| HEX$(*n*) | Converte *n* in una stringa esadecimale |
| INKEY$ | Legge un carattere da tastiera |
| INPUT$(*n*, #*f*) | Legge *n* caratteri dal file *f* |
| MKI$(*x*), MKS$(*x*), MKD$(*x*) | Converte *x*, nella precisione indicata, ad una stringa di lunghezza adeguata |
| OCT$(*n*) | Converte *n* in una stringa ottale |
| SPC(*n*) | Inserisce *n* spazi in un'istruzione PRINT o LPRINT |
| STR$(*x*) | Converte *x* in una stringa |
| TAB(*n*) | Compie uno spostamento alla posizione *n* in un'istruzione PRINT o LPRINT |
| TIME$ | Restituisce l'ora corrente di sistema |
| VARPTR$(*v*) | Restituisce una stringa di tre byte contenente il tipo della variabile ed il suo indirizzo in memoria |

### Funzioni di tipo stringa che operano su stringhe

Alcune funzioni-stringa hanno anche come argomento una stringa, ad esempio:

| | |
|---|---|
| `PRINT LEFT$(NOME$(Q),10)` | Visualizza i dieci caratteri più a sinistra dell'elemento dell'array di tipo stringa NOME$(Q) |

Le funzioni come LEFT$ sono usate per estrarre parti specifiche da una stringa data; funzioni simili a questa sono MID$ (per la parte centrale) e

RIGHT$ (per la parte più a destra); un'altra funzione spesso utile è STR$, di cui mostriamo un esempio:

| | |
|---|---|
| `PRINT STR$(N)` | Visualizza la rappresentazione in formato stringa del valore della variabile numerica N. Se N=4923, STR$(N) sarà uguale alla stringa "4923". |

### Funzioni di tipo stringa che operano su numeri

Altre funzioni di tipo stringa utilizzano un input numerico per produrre come risultato una stringa.

| | |
|---|---|
| `PRINT CHR$(CODICE)` | Visualizza il carattere il cui codice ASCII corrisponde al valore della variabile CODICE |
| `PRINT SPACE$(SPAZIO)` | Visualizza una stringa di spazi bianchi, la cui lunghezza è determinata dal valore della variabile SPAZIO |

La funzione CHR$, come già accennato nel Capitolo 4, serve per generare caratteri ASCII che non compaiono in tastiera; la funzione SPACE$ è usata, ad esempio, per aggiungere un determinato numero di spazi bianchi in un'istruzione PRINT, al fine di migliorarne l'apparenza.

## FUNZIONI DEFINITE DALL'UTENTE

Potete inoltre definire funzioni personalizzate per mezzo dell'istruzione DEF FN, la cui sintassi è la seguente:

DEF FN*nome* [(*argomento* [*argomento*,]...)]=*espressione*

Il parametro *nome* può essere un qualsiasi nome di variabile valido a vostra scelta; dopo aver definito la funzione, la richiamerete premettendo le lettere FN al nome. Potreste perciò scrivere, per la funzione REVERSE:

```
300 T=FNREVERSE(X)
```

I parametri *argomento* sono gli argomenti della funzione che devono essere definiti esplicitamente ogni volta che la funzione viene chiamata; *espressione* indica come la funzione opera sugli argomenti.

Supponiamo che voi vogliate una funzione che calcola l'area di un cerchio dato il raggio; assegnamo alla variabile PI# il valore di pi greco prima della definizione della funzione. Chiamiamo la funzione FNAREA ed usiamola in questo modo:

```
20 DEF FNAREA(X)=PI#*X*X
     .
     .
     .
450 SUPAREA=ALTEZZA*2*PI#*RAG*2*FNAREA(RAG)
     .
     .
     .
```

Quando viene chiamata la funzione FNAREA, il BASIC esamina la definizione e rimpiazza la variabile fittizia (in questo caso X) con l'argomento indicato nella chiamata della funzione (in questo caso, la variabile RAG).
Le funzioni definite dall'utente possono avere come risultato sia numeri che stringhe; il nome di una data funzione, però, dev'essere valido per il tipo di variabile che deve restituire: se la funzione ha come risultato una stringa, il nome della funzione deve terminare con il simbolo $, altrimenti il BASIC visualizzerà un messaggio d'errore TYPE MISMATCH.
Una funzione definita dall'utente può anche richiamare se stessa; è presente un meccanismo che impedisce che la chiamata sia ripetuta più di un certo numero di volte, in qual caso si originerebbe un messaggio d'errore OUT OF MEMORY.

## 5.6 Le istruzioni

Le istruzioni sono i mattoni con cui viene costruito un programma: un'istruzione BASIC è composta da una parola speciale (una parola chiave) e spesso da un'espressione o una funzione. Potete usare istruzioni per inserire dati, eseguire calcoli, manipolare stringhe o numeri e visualizzare risultati; esistono anche istruzioni che prendono decisioni basandosi su calcoli precedenti o sui valori di variabili ed espressioni; un ulteriore, importante, compito delle istruzioni è di controllare l'esecuzione del programma.
Un'istruzione BASIC a volte viene indicata come comando. La differenza tra istruzione e comando è che l'istruzione è tipicamente usata in modo indiretto, mentre il comando agisce più spesso in modo diretto. In ogni caso, la maggior parte dei comandi e delle istruzioni può essere utilizzata in entrambi i contesti.

## ISTRUZIONI DI COMMENTO

Una delle più importanti componenti di un programma è quella che viene chiamata istruzione di commento: questa non viene eseguita dal computer, ma serve all'utente del programma a comprendere che cosa il programma stesso stia compiendo in un determinato punto.
Sono due i modi in cui potete scrivere le istruzioni di commento: facendo precedere il testo del commento dalla parola chiave REM (*Remark*= commento) o dal carattere ' (apice). Le due linee che seguono, ad esempio, sono equivalenti:

```
340 REM Inizia l'ordinamento con W=10

340 ' Inizia l'ordinamento con W=10
```

I commenti possono essere inseriti anche alla fine di una linea di programma, usando però, solo l'apice:

```
340 W=10 ' Inizia l'ordinamento con W=10
```

Quando si aggiungono commenti nei programmi è buona abitudine essere concisi ma chiari.

## ISTRUZIONI DI ASSEGNAMENTO

Un programma BASIC può ricevere dati da due diverse fonti: o da sorgenti esterne, come l'operatore o una periferica, oppure dall'interno del programma stesso. Del primo tipo di sorgente, che viene solitamente chiamato con il generico nome di I/O (Input/Output) tratteremo tra breve; parliamo ora di come i dati possono essere definiti e come si può accedervi dall'interno di un programma, usando quelle che vengono chiamate istruzioni di assegnamento.

### Istruzione LET

Il più semplice modo per definire il valore di una variabile è di usare l'istruzione LET, che assegna un valore numerico o una stringa ad una variabile.
Un esempio potrebbe essere:

```
20 LET ROTAXO%=300 ' Inizializza ROTAXO%
```

L'uso della parola chiave LET è facoltativo, cosicché lo stesso risultato si può ottenere con l'istruzione:

```
20 ROTAXO%=300 ' Inizializza ROTAXO%
```

Quando l'assegnamento riguarda dei numeri, se la precisione della variabile differisce da quella della costante, quest'ultima è convertita alla precisione della variabile. Ad esempio, l'istruzione:

```
230 I%=3.5411
```

assegna a I% il valore 4.
In un'istruzione LET possono comparire anche delle espressioni del tipo:

```
340 U#=T/3+FATTCORR
350 FLAG= (NOT A AND B) OR (C AND D)
```

Per mezzo dell'istruzione di assegnamento possono essere definite anche delle variabili di tipo stringa:

```
450 PREAMBOLO$="Ottantasette anni fa"
```

Se tentate di assegnare una stringa ad una costante numerica o viceversa, il BASIC visualizzerà un messaggio d'errore TYPE MISMATCH.

NOTA: vedi le funzioni STR$ e VAL, nell'Appendice A, per la conversione tra rappresentazione numerica e in formato stringa.

### Istruzioni DATA e READ

Potete usare le istruzioni DATA e READ se nel programma devono essere utilizzate molte variabili il cui valore è una costante definita. La prima serve per predisporre una lista di costanti (sia numeriche che di tipo stringa) che verranno poi utilizzate dalla seconda. Le due istruzioni seguenti creano la lista visualizzata qui di seguito:

La lista prodotta è:

```
130 DATA 2.1,3.2,4.3,5.4,6.5,7.6,8.7 }————→ 2.1
140 DATA 9.8,0.9,1                   }      3.2
                                            4.3
                                            5.4
                                            6.5
                                            7.6
                                            8.7
                                            9.8
                                            0.9
                                            1
```

Le istruzioni READ assegnano alle variabili le costanti dell'istruzione DATA sulla base di un rapporto uno-a-uno; dopo che alla prima variabile è stato assegnato il primo valore della lista DATA, il BASIC sposta avanti di una posizione in questa lista un puntatore ai dati, in modo che alla variabile incontrata, sia nella stessa istruzione READ che in un'altra, venga assegnato il valore successivo. Supponiamo che siano utilizzate le due istruzioni insieme alle due precedenti istruzioni DATA:

```
200 READ T
210 READ Z
```

Il BASIC assegnerà il valore 2.1 a T ed il valore 3.2 a Z.
Non è importante l'ordine relativo all'interno del programma tra la serie di istruzioni DATA e la serie di istruzioni READ: ogni volta che il BASIC deve eseguire un READ, cerca lungo tutto il programma l'istruzione DATA che contiene il successivo dato da leggere.
Se in un programma le istruzioni READ cercano di leggere più dati di quelli definiti in precedenza, verrà visualizzato un messaggio d'errore OUT OF DATA; se invece sono presenti più dati di quelli richiesti dalle istruzioni READ, i termini in eccesso saranno trascurati.
Le costanti di tipo stringa non devono essere delimitate dalle virgolette all'interno delle istruzioni DATA, a meno che non contengano virgole, punti e virgola o un certo numero di spazi "significativi" all'inizio o alla fine.
Quando in un'istruzione DATA sono mescolati dati di tipo stringa e numeri, il tipo di costante dei DATA e il relativo nome della variabile dei READ devono essere in accordo:

```
40 READ R,R$
       .
       .
       .
400 DATA 594,La rosa
```

L'istruzione READ assegnerà alla variabile R il valore 594 ed alla variabile R$ la stringa "La rosa".

### Istruzione RESTORE

L'assegnamento sequenziale di costanti alle variabili per mezzo delle istruzioni READ e DATA può essere ripristinato con l'istruzione RESTORE. Quando il BASIC esegue RESTORE, il puntatore ai dati viene riportato indietro fino all'inizio della lista. La successiva istruzione READ, perciò, avrà lo stesso effetto della prima che compare nel programma, come potete vedere:

```
600 DATA 534,594,4.33,92.4444,88
        .
        .
        .
120 READ X
130 READ Y
140 READ Z
150 RESTORE
160 READ Q
```

Le linee dalla 120 alla 140 assegnano rispettivamente i valori 534 a X, 594 a Y e 4.33 a Z. L'istruzione RESTORE riporta il puntatore ai dati all'inizio della lista DATA: in questo modo a Q viene assegnato il valore 534.
Questa istruzione consente anche l'uso del parametro opzionale *linea*: se questo viene indicato, la successiva istruzione READ leggerà il primo termine che compare nell'istruzione DATA alla linea indicata. Il programma:

```
10 PRINT "Prima di RESTORE : ";:GOSUB 50
20 RESTORE 90 'RESTORE alla linea 90
30 PRINT "Dopo RESTORE : ";:GOSUB 50
40 END
50 READ D1$,D2$,D3$
60 PRINT D1$,D2$,D3$
```

```
70 RETURN
80 DATA "Primo"
90 DATA "Secondo"
100 DATA "Terzo"
110 DATA "Quarto"
```

visualizzerà le due righe:

```
Prima di RESTORE : Primo        Secondo      Terzo
Dopo RESTORE : Secondo     Terzo     Quarto
```

**Istruzione CLEAR**

In realtà l'istruzione CLEAR non è una tipica istruzione di assegnamento, ma pure assegna dei valori ad alcune variabili all'interno di un programma BASIC, nel senso che quando il BASIC esegue

```
CLEAR
```

a tutte le variabili numeriche del programma viene assegnato il valore 0 e alle variabili di tipo stringa il valore "stringa nulla" (cioè viene cancellata la stringa precedente). Ecco un esempio:

```
Ok
TEST$="Volpe marrone vivace...."
Ok
TEST=9
Ok
PRINT TEST$ TEST
 Volpe marrone vivace....  9
Ok
CLEAR
PRINT TEST$ TEST
 0
```

L'istruzione CLEAR può essere usata anche per lasciare più spazio per programmi in linguaggio macchina e per riservarne allo stack, quando necessario.

## 5.7 Istruzioni di controllo del flusso del programma

In circostanze normali il BASIC esegue le linee di programma nell'ordine indicato dai numeri di linea: quest'ordine può essere modificato, però, usando le cosiddette *istruzioni di flusso* del programma.
Quando il BASIC trova una di queste istruzioni, l'esecuzione del programma salta ad un'altra linea, non posta in sequenza con le precedenti, se sono verificate alcune condizioni. Le condizioni, naturalmente, sono indicate nell'istruzione di flusso. Le istruzioni di flusso sono usate per:

— Eseguire una parte del programma più volte, ogni volta con dati diversi
— Eseguire diverse parti del programma in funzione di alcune condizioni
— Utilizzare più volte lo stesso gruppo di istruzioni senza doverlo riscrivere.

**L'ISTRUZIONE GOTO**

L'istruzione GOTO fa sì che il BASIC salti alla linea il cui numero viene indicato nel corpo dell'istruzione ogni volta che l'istruzione stessa viene eseguita: provoca, perciò, un salto incondizionato. Ad esempio:

```
100 ...
110 GOTO 180
120 ...
130 ...
140 ...
150 ...
160 ...
170 ...
180 ...
190 ...
```

A causa dell'istruzione GOTO nella linea 110, l'esecuzione passa dalla linea 110 alla 180.
L'istruzione GOTO può causare anche salti all'indietro, come vedete:

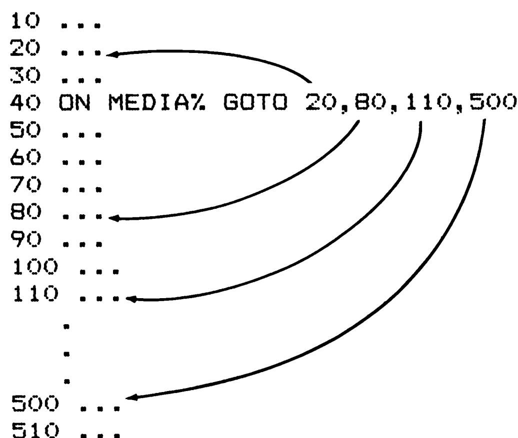
```
200 ...
210 ...
220 ...
230 ...
240 ...
250 ...
260 ...
270 GOTO 210
280 ...
```

## L'ISTRUZIONE ON-GOTO

Questa istruzione permette di saltare ad una tra diverse linee a seconda del valore di un'espressione numerica inserita nell'istruzione. La sintassi dell'istruzione ON-GOTO è

ON *espressione* GOTO *linea* [,*linea*]...

Il BASIC calcola l'*espressione*, arrotonda il risultato ad intero se necessario e salta alla linea col numero corrispondente al valore dell'*espressione*. Se l'*espressione* valesse 5, perciò, l'esecuzione passerebbe alla linea il cui numero occupa la quinta posizione nell'elenco. Se il risultato dell'*espressione* fosse maggiore del numero dei numeri di linea, o fosse uguale a zero, l'esecuzione continuerebbe alla linea di programma successiva. Un messaggio d'errore ILLEGAL FUNCTION CALL (Chiamata illegale ad una funzione) viene visualizzato se il valore dell'espressione risulta negativo o maggiore di 255.
Il flusso del programma creato da un'istruzione ON-GOTO può essere illustrato come segue:




Quando esegue la linea 40, il BASIC controlla il valore della variabile MEDIA%: se questa vale 1, la linea successiva ad essere eseguita sarà la 20; se vale 2, la 80; se vale 3, la linea 110; se vale 4, la linea 500; se invece MEDIA% è maggiore di 4 o uguale a 0, verrà eseguita per prima la linea 50.

## LOOP DI PROGRAMMA

Un *loop* (o ciclo) all'interno del programma consente di usare la stessa parte di programma più volte, di solito con dati differenti. Se voleste caricare in memoria un array di 100 elementi che contiene i numeri da 1 a 100, dovreste scrivere cento istruzioni di assegnamento:

```
10 CONT(0)=1
20 CONT(1)=2
      .
      .
      .
990 CONT(98)=99
1000 CONT(99)=100
```

Un metodo molto più veloce che vi consente di ottenere esattamente lo stesso risultato è di creare un loop usando le istruzioni FOR e NEXT: queste istruzioni fanno sì che la parte di programma compresa tra di esse venga eseguita un certo numero di volte, in accordo con una variabile il cui valore viene incrementato o decrementato ad ogni passo del loop. Per riempire l'array dell'esempio precedente e contemporaneamente visualizzarlo provate ad utilizzare il seguente programma:

```
10 DIM CONT(100)
20 FOR I=0 TO 99
30 CONT(I)=I+1
40 PRINT "QUESTO E' IL PASSO NUMERO " CONT(I)
50 NEXT
60 END
```

Le linee 20, 30, 40 e 50 costituiscono il loop: l'istruzione FOR fa in modo che l'indice I sia incrementato di uno ogni volta che si incontra l'istruzione NEXT, cioè ad ogni passo del loop, partendo da I=0 al primo passo. Quando I raggiunge il valore 99, cioè il limite specificato nell'istruzione FOR, il loop viene eseguito l'ultima volta e l'esecuzione continua poi dalla linea 60.
Il valore della variabile-indice viene confrontato con l'intervallo di variabilità definito nell'istruzione FOR dopo che la variabile è stata incrementata; la forma più generale in cui compare l'istruzione è:

FOR *variabile*=*x* TO *y* [STEP *z*]

      :

NEXT [*variabile*] [,*variabile*]...

Il parametro facoltativo STEP $z$ definisce il valore dell'incremento dell'indice: se $z$ non è specificato, viene assunto l'incremento di default che vale 1. Il valore del passo (STEP) può essere sia positivo che negativo: se è positivo, il loop termina quando il valore dell'indice è maggiore di $y$; se è negativo, termina quando il valore dell'indice è minore di $y$.
Se utilizzate un passo negativo, assicuratevi che il valore di $x$ non sia minore o uguale al valore di $y$, nell'istruzione FOR: in caso contrario il loop verrà eseguito solo una volta; nello stesso modo, verrà eseguito una volta sola se il passo è positivo ed $x$ è maggiore di $y$.
Per comprendere appieno il funzionamento dell'istruzione FOR...NEXT, provate a ripetere più volte il programma precedente utilizzando diversi valori per il passo STEP $z$. Osservate che possono anche essere utilizzati numeri reali all'interno dell'istruzione FOR, ma l'esecuzione sarà decisamente più veloce se i valori sono tutti interi.

## LOOP ANNIDATI

È possibile inserire dei loop formati con le istruzioni FOR e NEXT, in altri loop, in modo da creare loop annidati:

```
10 FOR A=1 TO 20
20 ...
30 ...
40   FOR XCONT=0 TO 99
50        FOR YCONT=0 TO 9
60             FOR ZCONT=0 TO 9
70             .
               .
               .
120  NEXT ZCONT,YCONT,XCONT
130 ...
140 ...
150  FOR BETA=0 TO 360 STEP 45
160 ...
               .
               .
               .
300  NEXT BETA
310 ...
320 NEXT A
```

Osservate le relazioni tra i vari loop, come indicate dalle frecce: ad esempio, notate che ogni volta che viene eseguito il loop caratterizzato dall'indice XCONT, il loop YCONT viene eseguito 10 volte, mentre il loop ZCONT addirittura 100 volte.

Il BASIC consente alcune piccole variazioni per quanto riguarda l'istruzione NEXT. La linea 120

```
120 NEXT ZCONT,YCONT,XCONT
```

avrebbe potuto essere sostituita da:

```
120 NEXT ZCONT
121 NEXT YCONT
122 NEXT XCONT
```

o più semplicemente da:

```
120 NEXT
121 NEXT
122 NEXT
```

Bisogna prestare molta attenzione a che la fine di un loop più esterno non si trovi prima della fine di un loop più interno; in questo caso, come potete constatare, il loop non può essere eseguito:

```
30 FOR R=50 TO 0 STEP -5
40 ...
50 FOR T=23 TO 29
60 ...
        .
        .
        .
130 NEXT R
        .
        .
        .
170 NEXT T
```

Questa struttura non è consentita perché la fine del loop interno T si trova dopo la fine del loop esterno R.
Provate a far eseguire questo programma, per comprendere il funzionamento dei loop annidati:

```
10 FOR MAIN=0 TO 4
20   FOR X=0 TO 10
30     FOR Y=0 TO 10
40       PRINT CHR$(1);" ";
50     NEXT Y
60   NEXT X
```

```
70   PRINT "BUON DIVERTIMENTO"
80   FOR Z=0 TO 4
90     BEEP
100  NEXT Z
110 NEXT MAIN
```

Notate come il loop tra le linee 80 e 100 in sostanza abbia l'effetto di aumentare la durata dell'istruzione BEEP: il suono che ne risulta sembra infatti un lungo fischio, più che cinque separati, come in realtà è. Notate anche che i loop annidati sono scritti con una rientranza relativa alla loro posizione, per mezzo dell'inserimento di alcuni spazi bianchi tra il numero di linea e l'istruzione vera e propria: questo non ha alcun effetto sull'esecuzione del programma, ma lo rende più leggibile, specialmente quando contiene strutture molto complesse.
I loop vengono usati anche per creare dei ritardi nel programma: per ritardi più lunghi potete usare loop annidati, che rappresentano una struttura compatta per eseguire un numero più alto di istruzioni. Per apprezzare maggiormente quanto detto, osservate che nel seguente segmento di programma, il numero totale di esecuzioni del loop è 1000+1000=2000:

```
300 FOR X=0 TO 1000
310 NEXT
320 FOR Y=0 TO 1000
330 NEXT
```

mentre, nel seguente:

```
300 FOR X=0 TO 1000
310 FOR Y=0 TO 1000
320 NEXT Y,X
```

il numero totale di passi è uguale a 1000×1000=1000000.
Per verificare quanto questa esecuzione sia veloce in tempo reale, provate ad utilizzare alcuni loop annidati come loop di ritardo, e modificate l'intervallo di variabilità degli indici. Potreste iniziare con il seguente programma:

```
10 PRINT "INIZIO DEL TEST"
20 FOR X=0 TO 10
30    FOR Y=0 TO 10
40 NEXT Y,X
50 BEEP
60 PRINT "FINE DEL TEST, CON X=" X " E Y=" Y
70 END
```

Quando apparirà sullo schermo il messaggio finale, vi accorgerete che il valore degli indici è maggiore di uno rispetto al valore massimo che avete specificato nelle rispettive istruzioni FOR. Ciò accade anche qui, perché l'istruzione NEXT incrementa gli indici prima di controllarne il valore.

## LE ISTRUZIONI WHILE E WEND

In modo molto simile alle istruzioni FOR e NEXT, anche le istruzioni WHILE e WEND possono essere usate per creare un loop; la sintassi di queste istruzioni è la seguente:

WHILE *espressione*
WEND

In questo caso, il loop che si trova tra le due istruzioni viene eseguito finché l'*espressione* dell'istruzione WHILE non risulta vera (cioè diversa da 0). Anche i loop WHILE-WEND possono essere annidati come i loop FOR-NEXT.
Questa struttura risulta molto utile quando vi serve ripetere un'operazione un certo numero di volte e il numero delle iterazioni dipende dal valore di una variabile che viene elaborata all'interno del loop stesso.

## LE ISTRUZIONI GOSUB E RETURN

Se la stessa azione dev'essere ripetuta in diversi punti del programma, l'istruzione più utile è GOSUB: ogni volta che incontra una linea contenente un'istruzione di questo tipo, il BASIC conserva il numero della linea e poi salta alla linea il cui numero è specificato nell'istruzione GOSUB. Quando poi incontra un'istruzione RETURN, il BASIC controlla il numero di linea conservato e ritorna alla linea successiva al GOSUB. La linea indicata dall'istruzione GOSUB è l'inizio di quella che viene detta *subroutine*: una subroutine è "chiamata" dal programma principale proprio per mezzo dell'istruzione GOSUB e la sua esecuzione termina non appena si incontra un'istruzione RETURN. Proprio per questo tutte le subroutine devono terminare con RETURN in modo che il BASIC possa riconoscerne la fine ed eseguire correttamente il programma, altrimenti l'esecuzione continuerebbe al di là delle istruzioni previste provocando un errore.
Per spiegare meglio l'uso delle subroutine vi proponiamo il seguente programma:

```
10 A$="PRIMA"
20 B$="@*@"
30 CONT=1
40 GOSUB 100
50 A$="SECONDA"
60 B$="*@*"
70 CONT=2
80 GOSUB 100
90 END
100 ' ****QUESTA E' LA SUBROUTINE PRINCIPALE****
110  PRINT  "QUESTA  E' LA " A$ " ESECUZIONE
DELLA  ROUTINE PRINCIPALE"
120 GOSUB 170
130 CONT=CONT^3
140 GOSUB 170
150 PRINT " FINE DELLA " A$ " ESECUZIONE"
160 RETURN
170 ' ****QUESTA E' UNA SUBROUTINE ANNIDATA****
180 BEEP
190 PRINT "    QUESTA E' LA SUBROUTINE ANNIDATA"
B$
200 PRINT "CONT =" CONT
210 RETURN
```

Il corpo principale del programma è rappresentato dalle linee che vanno dalla 10 alla 90; la subroutine chiamata dal programma principale inizia alla linea 100 e termina alla linea 160, mentre una seconda subroutine, chiamata dalla prima, è composta dalle linee che vanno dalla 170 alla 210. Osservate che in questo esempio compare l'istruzione END nella linea 90: in questo caso l'istruzione è assolutamente necessaria, perché la subroutine segue immediatamente il corpo principale del programma; se non fosse stata inserita l'istruzione END, l'esecuzione non sarebbe terminata alla linea 90, ma sarebbe proseguita di nuovo dalla linea 100 in avanti.

Questo esempio serve anche a dimostrare come le subroutine possano essere annidate, proprio come i loop; in linea di principio le subroutine possono essere annidate quante volte è necessario, ma oltre ad un certo punto il BASIC non potrà più ricordare tutti gli indirizzi di ritorno (RETURN). Vedremo nel Capitolo 8 qualche sistema per evitare l'inconveniente.

### GOSUB calcolato

Il GOSUB calcolato, cioè l'istruzione ON-GOSUB, agisce esattamente come l'istruzione ON-GOTO, con la sola differenza che provoca la chiamata di una subroutine e quindi il ritorno dell'esecuzione alla linea immediatamente seguente l'istruzione ON-GOSUB. Eccone un esempio:

```
200 ON DIRECT GOSUB 1000,1100,1200
210 ...
     .
     .
1000 ' SUBROUTINE PER DIRECT=1
     .
     .
1050 RETURN
1100 ' SUBROUTINE PER DIRECT=2
     .
     .
1150 RETURN
1200 ' SUBROUTINE PER DIRECT=3
     .
     .
1250 RETURN
```

Quando la variabile DIRECT vale 1, la linea 200 chiamerà la subroutine che inizia alla linea 1000, così come quando DIRECT è uguale a 2 viene chiamata la subroutine di linea 1100 e quando DIRECT è uguale a 3, la subroutine di linea 1200. Se invece DIRECT è uguale a 0 o maggiore di 3, non verrà eseguita nessuna delle subroutine, ma la linea 210. Proprio come per l'istruzione ON-GOTO, il valore dell'espressione che compare in ON-GOSUB non può essere negativo o superiore a 255, altrimenti viene visualizzato il messaggio d'errore ILLEGAL FUNCTION CALL.

## L'ISTRUZIONE IF-THEN-ELSE

Le istruzioni ON-GOTO e ON-GOSUB sono un esempio di esecuzione condizionata, poiché l'azione intrapresa da queste istruzioni dipende dal valore di una certa espressione. Un altro tipo di istruzione che esegue azioni condizionate è l'istruzione IF-THEN-ELSE che, come abbiamo visto in alcuni esempi precedenti, serve per saltare ad una determinata linea di programma se alcune condizioni sono verificate.
Questa istruzione, la cui sintassi è

IF *espressione* [,] THEN *proposizione* [[,] ELSE *proposizione*]

controlla il valore logico dell'*espressione*: se è vera, viene eseguita la prima *proposizione*, oppure, se non viene indicata l'opzione ELSE, la prima linea successiva. Tutte le virgole che appaiono nella sintassi sono facoltative: servono semplicemente ad aumentare la leggibilità della frase. Le proposizioni indicate possono essere sia istruzioni che numeri di linea.

Un esempio:

```
520 IF TEST>100 THEN TEST=0:PRINT "SOVRACCARICO"
:GOTO 900
530 ...
```

Se la variabile TEST è maggiore di 100, viene azzerata, viene visualizzato un messaggio e l'esecuzione riprende dalla linea 900; altrimenti, se cioè TEST è minore o uguale a 100, la linea 530 sarà la successiva ad entrare in esecuzione.
Un altro esempio:

```
1000 IF LEFT$(Z$,1)="Q" OR LEFT$(Z$,1)="q" THEN
5000 ELSE RIPETI=1
1010 ...
```

In questo caso, se il primo carattere di Z$ è una Q (maiuscola o minuscola) nella parte precedente del programma, l'esecuzione passerà alla linea 5000, altrimenti la variabile RIPETI sarà posta uguale a 1, ed eseguita la linea 1010. Osservate come la funzione di tipo stringa LEFT$(Z$,1) dia il primo (cioè il più a sinistra) carattere della stringa Z$.
Il BASIC accetterebbe anche questa forma per la linea 1000 dell'esempio precedente:

```
1000 IF LEFT$(Z$,1)="Q" OR LEFT$(Z$,1)="q" GOTO
5000 ELSE RIPETI=1
```

cioè con la parola chiave GOTO al posto di THEN.
Anche le istruzioni IF-THEN-ELSE possono essere annidate, come vediamo qui di seguito:

```
 .
 .
 .
200  IF CALDO=FREDDO THEN 300 ELSE IF ALTO=BASSO
THEN  PRINT "Sono tutto sottosopra, dammi una ma
no!!": GOTO 400 ELSE IF AVANTI=INDIETRO THEN 500
ELSE END
 .
 .
 .
```

Se CALDO è uguale a FREDDO, viene eseguita la linea 300, altrimenti la variabile ALTO viene confrontata con la variabile BASSO. Se queste sono

uguali, appare un messaggio e l'esecuzione prosegue dalla linea 400; se non sono uguali, AVANTI viene confrontato con INDIETRO; di nuovo, se sono uguali il programma salta alla linea 500, altrimenti, se tutti i test sono falsi, il programma termina.

**Come confrontare i numeri nell'istruzione IF-THEN-ELSE**

Il confronto tra due variabili reali (in precisione semplice o doppia) in un'istruzione IF-THEN-ELSE dev'essere effettuato con molta attenzione: infatti, a seconda del modo in cui i numeri sono memorizzati nel PC, la loro rappresentazione può essere leggermente differente da quella decimale visualizzata. Per evitare questo problema è meglio controllare la non-uguaglianza di due termini piuttosto che la loro uguaglianza; se questo non può essere fatto, per le variabili vi conviene controllare che questa sia uguale ad un valore dato più o meno un piccolo errore.
Se voleste confrontare il valore della variabile PEPE, in precisione semplice, con il valore 5490, dovreste usare la seguente sintassi dell'istruzione:

```
IF ABS(PEPE-5490)<1.0E-2 THEN ...
```

in cui si controlla se il valore assoluto della differenza tra la nostra variabile ed il valore scelto è minore di 0.01. Questo fattore è stato scelto perché i numeri in precisione semplice usano sì sette cifre per la memorizzazione, ma solo le prime sei sono accurate; ciò significa che in questo caso il BASIC visualizzerebbe i numeri compresi tra 5489.995 e 5490.004 come il numero, arrotondato, 5490. Perciò tutti questi numeri saranno considerati uguali a 5490, in quanto i nostri criteri di giudizio controllano solo che il valore di PEPE sia uguale a 5490 entro un errore di 0.01.

## 5.8 Le istruzioni di input/output

In ogni programma esiste almeno un punto in cui occorre realizzare una comunicazione tra il mondo esterno ed il PC: le istruzioni BASIC usate a questo scopo vengono chiamate istruzioni di input/output o I/O; in questo paragrafo parleremo dettagliatamente delle principali istruzioni di I/O.

**OUTPUT**

Il BASIC dispone di alcune istruzioni in grado di inviare dati dal PC al mondo esterno. Tra queste ci sono:

PRINT, WRITE, LPRINT, PUT, OUT, BEEP,
SOUND, PLAY, DRAW, CIRCLE, PAINT

Qui parliamo diffusamente solo della più semplice di queste istruzioni, PRINT; le altre, che vengono usate per applicazioni più complesse, verranno trattate in capitoli successivi.

**L'istruzione PRINT**

Come abbiamo già visto, l'istruzione PRINT permette ad alcune informazioni generate da un programma di essere visualizzate sullo schermo; inoltre, l'istruzione PRINT è corredata da parecchie opzioni che permettono di definire il formato dei dati visualizzati.
Innanzitutto, un'unica istruzione PRINT con più di un'espressione può visualizzare queste espressioni in differenti modi: se queste sono separate da almeno uno spazio bianco o da un punto e virgola, verranno visualizzate una dopo l'altra su un'unica riga di video. Se le variabili X, Y e Z hanno rispettivamente i valori 43.2, −5, e 90, l'istruzione

```
PRINT X Y;Z "Buffo"
```

produrrà come output la riga

```
43.2 -5  90 Buffo
```

Osservate come sia stato inserito uno spazio aggiuntivo tra i numeri −5 e 90 e come, in ogni caso, venga automaticamente aggiunto dopo ogni numero visualizzato; inoltre, davanti ai numeri positivi viene aggiunto uno spazio ulteriore e davanti ai numeri negativi il segno.
Se invece le espressioni vengono separate da una virgola all'interno dell'istruzione PRINT, la loro visualizzazione rispetterà le tabulazioni del video.

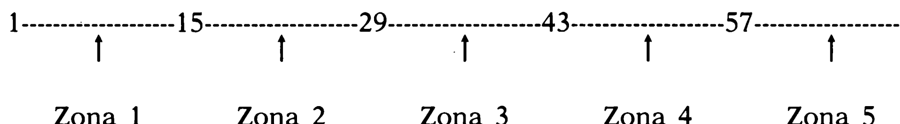
```
PRINT X,Y,Z,"Buffo"
```

avrà come output:

```
43.2          -5            90            Buffo
```

Come potete vedere, il BASIC divide la riga di schermo in diverse zone di stampa che permettono di organizzare più facilmente i dati in un formato ordinato. Ogni zona di stampa è lunga 14 o più colonne, a seconda della lunghezza del dato visualizzato. Il numero totale di zone di stampa per

riga di video dipende sia dal numero di caratteri che compongono ogni espressione sia dalla larghezza dello schermo (40 o 80 caratteri). Se tutte le espressioni di una data istruzione PRINT sono lunghe non più di 12 caratteri e lo schermo è predisposto per 80 colonne, ci sarà un totale di cinque zone di stampa per riga, come vedete:




In questo caso verranno visualizzate fino a cinque espressioni sulla stessa riga, ognuna delle quali giustificata a sinistra nella propria zona di stampa; in questo modo l'istruzione

```
PRINT 46.2, 11011024341, 67.5324, 2
```

produrrà il seguente output:

```
46.2          11011024341   67.5324       2
```

Se invece ogni espressione richiede più di 12 caratteri, allora verranno predisposte solo tre zone di stampa, o anche meno, per ogni riga di schermo. L'istruzione

```
PRINT "Questa e' l'ultima tacca -" , 23, 9
```

avrà come output:

```
Questa e' l'ultima tacca -    23            9
```

Infine, se nella stessa istruzione PRINT compaiono più espressioni di quelle che possono essere visualizzate su di un'unica riga di schermo, l'output sarà continuato nella riga seguente.

## INPUT

Il BASIC consente anche di inviare al computer informazioni provenienti dal mondo esterno: tra le istruzioni che permettono di eseguire questo compito troviamo:

INPUT, INKEY$, LINE INPUT, GET, INP

Per il momento discuteremo solo delle più semplici: l'istruzione INPUT e la funzione INKEY$.

**L'istruzione INPUT**

L'istruzione INPUT è in grado di svolgere due compiti: il principale è ricevere dati dalla tastiera e di assegnarne il valore alle variabili che compaiono nell'istruzione stessa; il secondo è di visualizzare un messaggio destinato all'operatore. La sintassi di questa istruzione è la seguente:

INPUT [;] ["*stringa prompt*";] *variabile* [,*variabile*]...

Quando il BASIC incontra un'istruzione INPUT in cui sia stata specificata la *stringa prompt*, la visualizza facendola seguire da un punto di domanda e attende che voi inseriate la risposta, seguita dal tasto ENTER, che significa che la linea è disponibile per essere letta.
Se invece questa stringa opzionale manca, viene visualizzato solo il punto interrogativo ed il BASIC di nuovo attende che voi abbiate terminato la vostra risposta. Inserendo una virgola dopo la *stringa prompt*, otterrete la soppressione del punto di domanda. Ecco un esempio di istruzione INPUT:

```
INPUT "Spessore, altezza e larghezza"; S,A,L
```

L'effetto risultante sarà:

```
Spessore, altezza e larghezza? 4.3,2,98
```

I valori 4.3, 2 e 98 sono solo degli esempi di valori che potreste inserire in risposta al prompt. Un altro esempio potrebbe essere:

```
INPUT "Nome e numero di matricola: " , N$,M
```

che ha come risultato:

```
Nome e numero di matricola: Gian Galeazzo Visconti, 3
```

Di nuovo, Gian Galeazzo Visconti e 3 sono solo esempi di come potreste rispondere.
Ogni termine scritto da tastiera deve corrispondere, come tipo, alla variabile corrispondente nell'istruzione INPUT, cioè a quella a cui dev'essere assegnato. Notate, comunque, che le stringhe che scrivete non devono essere comprese tra virgolette, ma i diversi termini devono essere separati tra di loro per mezzo di virgole.

Se il tipo dei termini battuti non corrisponde a quello delle variabili a cui devono essere assegnati, o se il numero di termini non è uguale al numero di variabili, il BASIC risponde con questo messaggio d'errore piuttosto oscuro: ?REDO FROM START. Ciò significa che l'input non è stato accettato, e la stringa di prompt viene ripetuta.
Se compare una sola variabile nell'istruzione INPUT, il premere solamente il tasto ENTER in risposta al prompt fa sì che a questa variabile venga assegnato il valore 0, se è una variabile numerica, o la stringa nulla, se è di tipo stringa.

### La funzione INKEY$

La funzione INKEY$ serve per leggere un singolo carattere da tastiera: se utilizzate questa funzione, il BASIC le assegna come valore la stringa contenente il carattere corrispondente al tasto che avete premuto; se non premete alcun tasto, le assegna la stringa nulla.
Per comprendere il funzionamento della funzione INKEY$, vi proponiamo il seguente programma:

```
10 A$=INKEY$
20 IF A$="" THEN PRINT "NESSUNA SCELTA" ELSE 100
30 GOTO 10
100 PRINT A$
110 FOR X=0 TO 1000
120 NEXT
130 GOTO 10
```

La funzione INKEY$ assumerà come valore la stringa nulla finché nessun tasto viene premuto, ma le sarà assegnata una stringa di uno o due caratteri se voi premete un tasto o una sequenza di tasti che invia da tastiera un codice ASCII esteso. Se INKEY$ è uguale ad una stringa di due caratteri, il primo codice sarà sempre 00 esadecimale; i codici ASCII estesi sono descritti nell'Appendice B.

Capitolo

# Uso avanzato del BASIC 6

Una parte essenziale di ogni programma è rappresentata dalle istruzioni di input/output che permettono al programma stesso di entrare in comunicazione con il mondo esterno; il BASIC dispone di parecchie risorse utilizzabili a questo fine. In questo capitolo impareremo alcune tecniche di uso dello schermo, della stampante e della tastiera; impareremo anche a scrivere programmi in grado di rispondere ad eventi esterni o ad errori che si verifichino durante l'esecuzione del programma.

## 6.1 Tecniche di visualizzazione su schermo

Lo schermo del PC può essere utilizzato in diversi modi per visualizzare in maniera semplice e compatta l'output di un programma.
I due modi di visualizzazione sono il modo *Text* e il modo *Grafico*. Il modo Text permette di porre sullo schermo ciascuno dei 256 caratteri e simboli disponibili; il modo Grafico permette di disegnare figure controllando individualmente ogni punto dello schermo. In questo capitolo ci dedichiamo allo studio del modo Text, mentre il modo Grafico sarà affrontato nel Capitolo 9.

### MODO TEXT

Il modo Text è disponibile su tutte le versioni del PC; alcune delle sue caratteristiche più complesse, come il colore e l'uso di finestre multiple, richiedono l'uso della scheda Colore/Grafica.

Nel modo Text, la procedura di base per produrre una videata (cioè tutto l'output che riempie completamente lo schermo) consiste nell'inizializzare lo schermo, muovere il cursore nella posizione scelta ed infine inviare i dati per mezzo delle istruzioni PRINT, PRINT USING e WRITE.

### Come inizializzare lo schermo

Prima di disporre qualcosa sullo schermo dovete inizializzarlo, definendone la larghezza e cancellando ogni eventuale informazione preesistente: tutto ciò viene compiuto dai due comandi WIDTH e CLS.
La larghezza dello schermo può essere, come già abbiamo visto, di 40 o di 80 caratteri: le due versioni del comando WIDTH perciò sono:

```
WIDTH 40
```

e

```
WIDTH 80
```

Quando il BASIC visualizza dei caratteri sullo schermo, aggiunge automaticamente un'istruzione di ritorno carrello e a capo se il numero di caratteri è maggiore di quello previsto per la singola riga dall'istruzione WIDTH; se la larghezza non viene specificata, viene assunta quella di default, cioè 80 caratteri. Se inserite un'istruzione che cambia la larghezza dello schermo all'interno di un programma, in quel punto lo schermo verrà anche cancellato.
Se volete iniziare il lavoro con lo schermo pulito, potete usare l'istruzione CLS, che abbiamo già incontrato nel Capitolo 4.
Quando l'istruzione CLS, viene eseguita, lo schermo al quale voi state inviando l'output viene ricoperto completamente con il colore del fondo ed il cursore viene portato nella posizione "home" (angolo in alto a sinistra dello schermo).

### Come posizionare il cursore

Controllando la posizione del cursore potete creare formati di output adatti per tabelle, mappe, diagrammi...
La posizione del cursore è definita da un numero di riga e dalla colonna all'interno di quella riga: le righe dello schermo del PC sono numerate da 1 a 25, mentre le colonne vanno da 1 a 40 oppure da 1 a 80, a seconda di come è stata predisposta la larghezza dello schermo. Le coordinate del cursore sono illustrate in Figura 6.1.

In ogni momento potete stabilire la posizione corrente del cursore utilizzando due funzioni: CSRLIN e POS(0); all'interno di un programma le funzioni verranno usate assegnandone il valore ad una variabile:

```
1090 VERT=CSRLIN : ORIZ=POS(0)
```

Questa istruzione serve a caricare il numero della riga in cui si trova il cursore nella variabile VERT e la colonna nella variabile ORIZ. Può essere utile conoscere l'effettiva posizione del cursore quando questo dev'essere spostato per compiere alcune operazioni e poi riportato nella posizione originale.
Potete anche posizionare il cursore in un qualsiasi punto dello schermo per mezzo dell'istruzione LOCATE, la cui sintassi è:

LOCATE [*riga*] [,[*colonna*] [,[*cursore*] [,[*start*] [,*stop*]]]]

I parametri *riga* e *colonna* stabiliscono la posizione del cursore; il parametro *riga* può essere una qualsiasi espressione numerica il cui valore sia compreso tra 1 e 25, mentre il parametro *colonna* può variare tra 1 e 40 (WIDTH=40) o tra 1 e 80 (WIDTH=80).
Il parametro *cursore* è un'espressione che se uguale a 0 cancella il cursore, mentre se viene posta uguale ad 1 lo fa riapparire.
I parametri *start* e *stop* sono espressioni numeriche il cui valore è compreso tra 0 e 31 e servono per controllare le dimensioni del cursore: i parametri si riferiscono alle righe di punti che compongono ogni singolo carattere; se è installata l'interfaccia IBM per monitor monocromatico e stampante parallela le righe sono numerate da 0 (la più in alto di ogni posizione di carattere) a 13; se invece è installata la scheda Colore/Grafica, la numerazione delle righe va da 0 (sempre dall'alto in basso) a 7. Il parametro *start* indica la prima riga all'interno della posizione del carattere che deve essere occupata dal cursore, ed il parametro *stop* indica l'ultima. Se il primo parametro è maggiore del secondo, si ottiene un cursore diviso in due parti.
Se uno qualsiasi dei parametri dell'istruzione LOCATE viene omesso, il BASIC utilizza gli ultimi valori definiti.
Nella Figura 6.1 potete vedere due esempi di come funziona l'istruzione LOCATE, con uno schermo largo 80 colonne:

LOCATE 1,1 Forma e aspetto del cursore invariati; il cursore si posiziona in home

LOCATE 20,59,1,0,13 Muove il cursore alla linea 20, colonna 59; cursore ON; il cursore occupa tutta la casella a disposizione

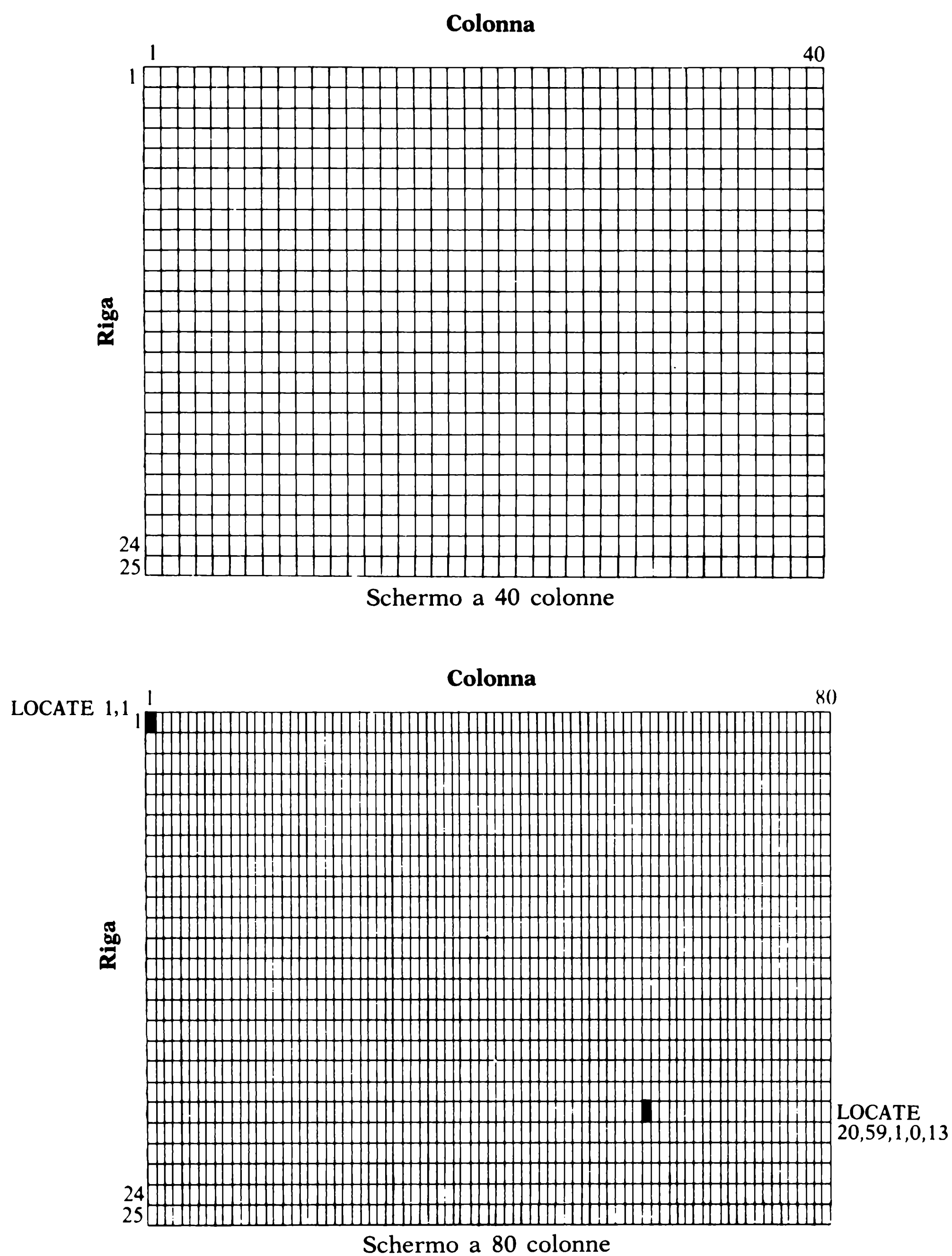



**Figura 6.1** Mappa dello schermo in modo Text

Infine esistono due funzioni che permettono di posizionare il cursore dall'interno di un'istruzione PRINT: sono le funzioni TAB(*n*) e SPC(*n*).
TAB(*n*) sposta il cursore fino alla colonna *n* della riga da esso occupata; *n* può variare da 1 a 40 o da 1 a 80, sempre a seconda della larghezza dello schermo. Se *n* è più piccolo del numero della colonna in cui si trova il cursore, questo si sposta nella linea seguente, alla colonna *n*.
La funzione SPC(*n*), invece, produce una stringa di *n* spazi bianchi: quando viene usata all'interno di un'istruzione PRINT, vengono visualizzati *n* spazi, cosa che equivale a spostare il cursore di *n* spazi a destra della posizione corrente.
Osservate la differenza tra le due funzioni: TAB(*n*) muove il cursore in una posizione assoluta dello schermo, mentre SPC(*n*) lo muove in una posizione relativa all'ultima occupata.
Provate ad eseguire queste istruzioni in modo diretto, per vedere come funzionano:

```
PRINT "Ci sono 5" SPC(5) "spazi tra le parole
5 e spazi."

PRINT "La prossima parola " TAB(5) " iniziera'
nella colonna 5 della linea successiva all'iniz
io di questa frase."
```

**Come inviare l'output allo schermo**

Ora che avete imparato a posizionare il cursore per mezzo dell'istruzione LOCATE o delle funzioni TAB e SPC, potete visualizzare sullo schermo i caratteri che volete nella posizione da voi scelta. L'istruzione PRINT serve per visualizzare un termine esattamente come voi l'avete scritto; l'istruzione PRINT USING, invece, dapprima formatta il dato a seconda delle direttive inserite nel corpo dell'istruzione, in modo da consentirvi di organizzare dati in tabelle o in strutture predefinite in modo molto agevole.
Per evitare che il cursore vada a capo dopo un'istruzione PRINT o PRINT USING, come accade normalmente, dovete terminare l'istruzione con un segno di punto e virgola; provate perciò il seguente programma:

```
WIDTH 40

LOCATE 10,10:PRINT "QUESTA E' LA RIGA 10":PRINT
"QUESTO  E' L'INIZIO DELLA RIGA 11"
```

Ora tornate alla larghezza di 80 colonne, riportate il cursore fino al bordo superiore dello schermo e battete:

```
LOCATE 15,10:PRINT "QUESTA E' LA RIGA 15";:PRINT
" E QUESTA E' ANCORA LA RIGA 15"
```

L'istruzione PRINT USING esegue una formattazione dei dati sulla base di una *maschera*, cioè una stringa contenente alcuni "caratteri di formato". Questi caratteri indicano il modo in cui ogni dato dev'essere visualizzato. La sintassi dell'istruzione è

PRINT USING X$; *lista di espressioni* [;]

La costante o variabile di tipo stringa X$ è la maschera; i caratteri di formato in X$ possono specificare il numero di caratteri che devono essere visualizzati per ogni espressione di tipo stringa, il numero di cifre da scrivere a destra o a sinistra della virgola e inoltre se un dato numerico debba essere preceduto o meno da caratteri alfabetici (es.: Lire). Per il momento consideriamo solo alcuni dei caratteri di formato. Nell'Appendice A sono elencati, tra gli altri, anche tutti i caratteri di formato e le loro funzioni.
Esaminiamo l'istruzione:

```
PRINT USING "\    \";T$,Y$,U$
```

La maschera qui è rappresentata da due backslash separati da quattro spazi: ciò significa che verranno visualizzati in questo modo solo i primi sei caratteri delle stringhe in output. Se abbiamo, ad esempio:

```
T$="Fee"
Y$="Fi"
U$="Fo"
```

la precedente istruzione PRINT USING produrrebbe come output la riga:

```
Fee   Fi    Fo
```

Osservate che anche se le stringhe hanno un numero di caratteri inferiore a quello indicato, sono visualizzati sei caratteri e quelli mancanti vengono aggiunti come spazi vuoti. D'altra parte, se avessimo:

```
T$="Liberte"
Y$="Egalite"
U$="Fraternite"
```

la stessa istruzione PRINT USING darebbe in output:

```
LibertEgalitFrater
```

Per vedere come l'istruzione PRINT USING può essere usata con i numeri, provate a meditare sul seguente problema: supponete di voler visualizzare dei numeri di diversa lunghezza (o precisione) su tre colonne. Potete incolonnare il punto decimale, ma tutti i numeri, in ogni colonna, devono avere diverso formato (lunghezza). Ecco un programma che vi mostra come si risolve il problema:

```
10 DATA 123.333,2,33.53,6376,4,53.3254,45,4,5.084
20 FOR I=0 TO 8
30 READ A(I)
40 NEXT I
50 WIDTH 80:LOCATE 10,1
60 FOR I=0 TO 8
70 PRINT USING "####.##";A(I);
80 I=I+1
90  PRINT  USING "      ##";A(I);  '6 spazi bianchi
fino al segno #
100 I=I+1
110  PRINT USING "     ###.###";A(I) '5 spazi bianchi
fino al segno #
120 NEXT I
130 END
```

In questo programma le maschere sono "####.##", "##", e "###.###": ogni simbolo # consente la visualizzazione di una cifra della costante o variabile numerica che segue la maschera (in questo caso, A(I)). Il segno . nella maschera indica dove deve essere posto il punto nella visualizzazione del numero.

È importante sottolineare che il numero di simboli # che compaiono a destra del punto determina il modo in cui i numeri verranno arrotondati prima della visualizzazione; notate anche che gli spazi che precedono i simboli # nella maschera verranno riportati esattamente sullo schermo.

## USO DEL COLORE CON SCHERMO IN MODO TEXT

Il colore può rendere decisamente più piacevole l'aspetto del vostro output: se impiegato correttamente può richiamare l'attenzione sulle parti più importanti della videata, facendole risaltare sul tutto.

Vi è indispensabile avere il supporto della scheda Colore/Grafica per usare il colore nei vostri testi: potete specificare sia il colore dei caratteri che quello del fondo, così come il colore del bordo che circonda la parte di schermo riservata ai testi.

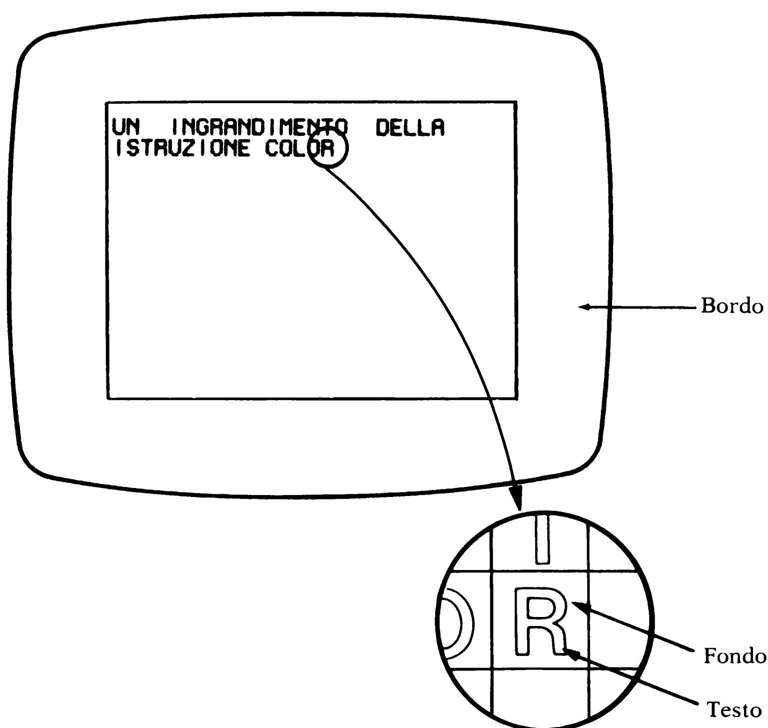
Potete però sfruttare alcune caratteristiche che verranno qui trattate, come la sottolineatura ed il lampeggio, anche se possedete solamente l'interfaccia per monitor monocromatico e stampante parallela.

**L'istruzione COLOR**

Nel modo Text il colore viene controllato per mezzo dell'istruzione COLOR, la cui sintassi è:

COLOR [*testo*] [,[*fondo*] [,*bordo*]]

Se possedete una scheda Colore/Grafica, il parametro *testo* definisce il colore di tutti i caratteri in output, il parametro *fondo* il colore dell'area tra i caratteri ed il parametro *bordo* il colore dell'area che contorna lo schermo.
Queste aree sono ben indicate nell'illustrazione che segue




I valori dei parametri dell'istruzione COLOR sono correlati ai colori come mostrato dalla Tabella 6.1; il valore relativo al testo può variare da 0 a 31: se è compreso tra 0 e 15 il colore sarà uno di quelli elencati; se invece è compreso tra 16 e 31, sarà ancora uno dei colori riportati, con la caratteristica aggiuntiva che il carattere stesso lampeggerà. In questo modo il valore 6 per il testo produrrà caratteri di color marrone, mentre

**Tabella 6.1** Valori dei colori per la scheda Colore/Grafica

| Numero | Colore | Numero | Colore |
|---|---|---|---|
| 0 | Nero | 8 | Grigio |
| 1 | Blu | 9 | Blu chiaro |
| 2 | Verde | 10 | Verde chiaro |
| 3 | Cyan | 11 | Cyan chiaro |
| 4 | Rosso | 12 | Rosso chiaro |
| 5 | Magenta | 13 | Magenta chiaro |
| 6 | Marrone | 14 | Giallo |
| 7 | Bianco | 15 | Bianco brillante |
| | | 16-31 | Come 0-15 con il lampeggio |

il valore 22 (6+16) produrrà caratteri lampeggianti di color marrone. Il valore del parametro del fondo può variare tra 0 e 7 (i colori tra 8 e 15 non sono permessi per il fondo), mentre il valore del parametro del bordo può variare tra 0 e 15.
Se invece usate un'interfaccia per monitor monocromatico e stampante parallela, i colori disponibili sono quelli riportati in Tabella 6.2 con i relativi valori del parametro. Se un normale televisore o un altro monitor sono collegati al PC per mezzo della scheda Colore/Grafica, l'effetto dei parametri dell'istruzione COLOR dipende in larga misura dallo schermo: in particolare alcuni schermi "sfocano" il carattere che così non appare nitido. Di solito ci si può rendere conto di questo fatto quando su di uno schermo regolato correttamente in contrasto e luminosità per un carattere a bassa intensità (es. testo=7) compaiono dei caratteri in un colore ad alta intensità (es. testo=15).

**Tabella 6.2** Valori dei colori per l'adattatore per monitor monocromatico e stampante parallela

| Numero | Testo | Fondo |
|---|---|---|
| 0 | Nero | Nero |
| 1 | Bianco sottolineato | Nero |
| 2 | Bianco | Nero |
| 3 | Bianco | Nero |
| 4 | Bianco | Nero |
| 5 | Bianco | Nero |
| 6 | Bianco | Nero |
| 7 | Bianco | Bianco |
| 8-15 | Alta intensità | |
| 16-31 | Lampeggio | |

### Caratteri non visibili

A volte potrebbe essere utile far scomparire dallo schermo alcuni caratteri: ad esempio potreste volere che una *password* (parola, solitamente segreta, usata dall'utente di un elaboratore per accedere al sistema ed alle sue risorse, che hà lo scopo di proteggere queste ultime da eventuali intrusioni indesiderate) usata da un operatore non compaia sul video per non essere letta da estranei.
A questo fine, potete far eseguire un'istruzione COLOR in cui il parametro *testo* abbia lo stesso valore del parametro *fondo*: questo renderà invisibili tutti i caratteri che appariranno successivamente. Un esempio:

```
         .
         .
         .
550 PRINT "Inserire la password - "
560 COLOR 0,0 ' Rende invisibili i caratteri
570 INPUT PASSWORD
580 COLOR 7,0 ' Rende i caratteri di nuovo visibili
590 PRINT "Grazie mille."
         .
         .
         .
```

## USO DELLA FUNZIONE SCREEN

La funzione SCREEN fornisce due tipi di informazioni a proposito di una determinata posizione dello schermo: essa identifica o il carattere che occupa la posizione oppure il colore ad essa assegnato.
La sintassi della funzione è:

X=SCREEN (*riga, colonna* [,*z*])

dove il tipo di informazione riportata dipende dal parametro *z*: se questo manca o è falso (uguale a 0), alla variabile X sarà assegnato il codice ASCII del carattere che occupa la posizione dello schermo indicata dai valori di *riga* e *colonna*; se invece *z* è presente e diverso da 0 (cioè è vero) la variabile X conterrà il colore attribuito a quella posizione.
L'attributo di colore permette di risalire sia al colore del testo che a quello del fondo così come al fatto se il carattere sia lampeggiante o meno. Le informazioni che riguardano la posizione possono essere dedotte dalla funzione SCREEN in questo modo:

— Colore del testo = X MOD 16
— Colore del fondo = (X MOD 128) \ 16
— Lampeggio se X>127

Le prime due formule hanno come risultato degli interi che corrispondono ai numeri riportati nella Tabella 6.2.
La funzione SCREEN è molto utile quando volete visualizzare un messaggio all'interno di una videata già esistente e poi ritornare alla versione originale. Il programma riportato in Figura 6.2 vi mostra come questo può essere fatto; osservate che questo programma presume che voi disponiate di una scheda Colore/Grafica.

```
10 'PROGRAMMA PER INSERIMENTO DI UN MESSAGGIO E
RIPRISTINO DELLO SCHERMO
20 '
30 ' Questo programma richiede la scheda Colore/Grafi
ca
40 '
50 DIM ATTRIBUTI(80) 'Dimensiona il buffer per gli at
tributi dello schermo
60 CLS:COLOR 16,2 ' Inizia su un nuovo schermo , con
colore nero lampeggiante su fondo verde
70 FOR T=0 TO 20  ' Loop per riempire lo schermo
80 PRINT TAB(T) "QUESTA E' LA VOLTA NUMERO " T " IN
CUI APPARE QUESTA LINEA"
90 NEXT T
100 COLOR 4,0  ' Usa il rosso su fondo nero per le is
truzioni
110 INPUT "QUAL E' IL MESSAGGIO CHE VOLETE INSERIRE "
;MESSAGGIO$
120 INPUT "IN QUALE RIGA E COLONNA VOLETE IL MESSAGGI
O";RIGAMESS,COLMESS
130 GOSUB 160 ' Va alla routine di inserimento del me
ssaggio
140 LOCATE 22,1:COLOR 4,0 ' Ripristina il cursore per
 ripetere le istruzioni
150 GOTO 110
160 '
170 ' ****Routine di inserimento del messaggio****
180 '
190 COLOR 4,0
200 STRINGAORIG$="" 'Inizializza la variabile di tipo
 stringa che deve contenere il testo originale
210 '
220 ' Questo loop memorizza tutti i caratteri dello
schermo che verranno coperti dal messaggio
230 '
240 FOR I=COLMESS TO (COLMESS + LEN (MESSAGGIO$))
250 ASCII=SCREEN(RIGAMESS,I) ' Prende il carattere da
lla posizione corrente
260 STRINGAORIG$=STRINGAORIG$+CHR$(ASCII) 'Aggiunge i
 caratteri al buffer del testo
```

(continua)

```
270 ATTRIBUTI(I-COLMESS)=SCREEN(RIGAMESS,I,1) ' Aggiu
nge gli attributi al buffer
280 NEXT I
290 LOCATE RIGAMESS,COLMESS ' Muove il cursore per vi
sualizzare il messaggio
300 PRINT MESSAGGIO$
310 GOSUB 430 ' Ritardo per mantenere il messaggio su
llo schermo
320 LOCATE RIGAMESS,COLMESS ' Muove il cursore per ri
pristinare il testo
330 '
340 ' Questo loop ripristina il testo originale
350 '
360 FOR I=COLMESS TO (COLMESS + LEN(MESSAGGIO$))
370 IF ATTRIBUTI (I-COLMESS)>127 THEN BLINK=16 ELSE B
LINK=0 ' Esamina se i caratteri originali sono lampeg
gianti
380 COLOR (ATTRIBUTI(I-COLMESS) MOD 16)+BLINK, (ATTRI
BUTI(I-COLMESS)/16),0 'Calcola l'attributo di colore
del carattere originale
390 Y$ = MID$(STRINGAORIG$,(I+1-COLMESS),1)
400 PRINT Y$; ' Riscrive il carattere originale
410 NEXT I
420 RETURN
430 '
440 ' *** Subroutine di ritardo ***
450 '
460 FOR RITARDO = 0 TO 800:NEXT RITARDO
470 RETURN
```

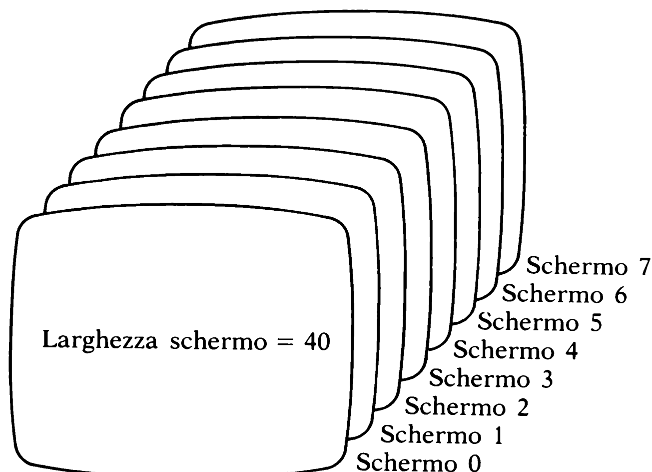
**Figura 6.2** Programma di inserimento di un messaggio e ripristino dello schermo

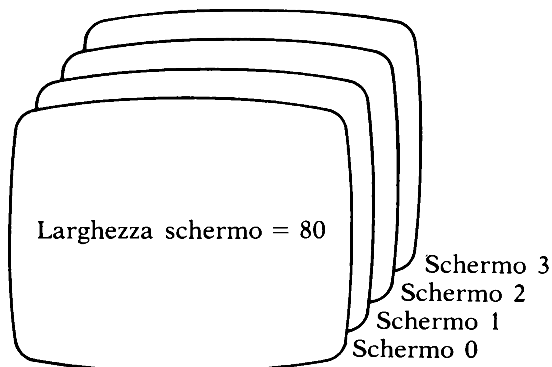
In questo esempio la funzione SCREEN è usata per riportare il testo precedente a ricoprire la zona in cui è apparso il messaggio: il testo è poi memorizzato e più tardi fatto ricomparire sullo schermo.

## PAGINE MULTIPLE

Nel modo Text la scheda Colore/Grafica è in grado di memorizzare più di una pagina di testo: ciò vi permette di visualizzarne una mentre ne state creando un'altra. Per spiegarci meglio, se voi sapete in precedenza che cosa deve apparire nella videata successiva potete prepararla su di una "pagina" nascosta mentre lo schermo vi mostra altre parti di testo. Al termine della preparazione la nuova videata, già definitiva, può essere visualizzata istantaneamente.
Se la larghezza dello schermo è di 40 colonne avete a disposizione fino a

8 pagine di schermo, mentre se la larghezza è di 80 colonne ne avrete solo 4. Questa illustrazione vi dà un'idea più chiara di cosa significhino diverse "pagine" di schermo:




**L'istruzione SCREEN**

L'istruzione SCREEN (ben diversa dalla funzione SCREEN vista in precedenza) permette di selezionare il modo Text o Grafico e consente l'uso dei colori; per mezzo di questa istruzione è anche possibile stabilire quale schermo usare per l'output successivo e quale invece visualizzare. La sintassi dell'istruzione SCREEN è la seguente:

SCREEN [*modo*] [,[*colore*] [,[*paginaa*] [,*paginav*]]]

Il parametro *modo* permette di scegliere il modo Text o uno tra due modi Grafici: è un'espressione numerica che può valere 0 (modo Text), 1 (modo Grafico a media risoluzione) oppure 2 (modo Grafico ad alta risoluzione); se lavorate in uno dei due modi Grafici, ricordate che avete a disposizione una sola pagina di schermo, a differenza di quanto accade in modo Text.
Il parametro *colore* abilita l'uso del colore; in modo Text un'espressione numerica falsa (cioè di valore 0) disabilita il colore mentre un'espressione vera (di valore diverso da 0) lo abilita; l'opposto accade in modo Grafica a media risoluzione, poiché un valore falso abilita il colore ed un valore vero lo disabilita. Nel modo Grafico ad alta risoluzione il parametro *colore* non ha alcun effetto, perché gli unici colori disponibili sono il bianco ed il nero.
Il parametro *paginaa* (pagina attiva) è una qualsiasi espressione numerica che determina a quale pagina di schermo sarà indirizzato il successivo output: se usate uno schermo a 40 colonne il valore del parametro può andare da 0 a 7, mentre con uno schermo ad 80 colonne deve variare tra 0 e 3.
Il parametro *paginav* (pagina visibile), analogamente al parametro *paginaa*, sceglie tra l'insieme delle pagine quella da visualizzare; notate come entrambi questi parametri siano efficienti solo in modo Text.

## Come alternare le pagine di schermo

Come abbiamo appena visto, potete visualizzare la pagina di schermo individuata dal parametro *paginav* e contemporaneamente inviare dati alla *paginaa*: quando quest'ultima è completa, potete visualizzarla semplicemente eseguendo un'altra istruzione SCREEN in cui avete cambiato il valore del parametro *paginav*. La pagina visualizzata in precedenza sarà salvata nella memoria della scheda Colore/Grafica.
Un punto da ricordare in questo scambio di pagine di schermo è che tutte le pagine utilizzano lo stesso cursore: se perciò voi state visualizzando del testo sulla pagina 0 ed il cursore si trova all'inizio della tredicesima riga e decidete di cambiare la pagina, la successiva istruzione PRINT visualizzerà il proprio contenuto a partire dalla riga 13. Dovete perciò memorizzare le coordinate del cursore se vi interessa tener conto della sua posizione su ogni diversa pagina di schermo. A questo scopo sono usate le funzioni CSRLIN e POS(0), come nell'esempio seguente:

```
       .
       .
       .
200 VERT=CSRLIN : ORIZ=POS(0) ' Memorizza le coordin
ate del cursore
```

```
210 SCREEN 0,1,1,1 ' Seleziona la pagina 1 (schermo
in modo Text) sia per output che per visualizzazione,
con scelta del colore
220 LOCATE 1,1 ' Posiziona il cursore nell'angolo in
alto a sinistra della nuova pagina
      .
      .
      .
300 SCREEN 0,1,0,0 ' Torna alla pagina 0 per output e
visualizzazione, con colore
310 LOCATE VERT,ORIZ ' Riporta il cursore nella posiz
ione originale
      .
      .
      .
```

Osservate che le due funzioni CSRLIN e POS(0) si riferiscono implicitamente alla pagina di schermo attiva. Lo 0 come argomento di POS è del tutto fittizio: non si riferisce, infatti, allo schermo 0.
Se dovete ripetere svariate alternanze di pagine di schermo, è più conveniente utilizzare per le coordinate del cursore variabili specifiche per ogni pagina, che potrebbero essere VERT1, ORIZ1; VERT2, ORIZ2 e così via.

## 6.2 Tecniche di output su stampante

Per utilizzare la stampante dovete aver collegato la vostra stampante parallela o seriale al PC rispettivamente per mezzo di una scheda di interfaccia parallela o seriale. L'interfaccia per il monitor monocromatico contiene anche un'interfaccia per stampante parallela; questa interfaccia consente l'uso di un grande numero di stampanti: faremo sempre riferimento però alla stampante a matrice di punti IBM e alle sue "cugine", le stampanti Epson MX-80 ed MX-100.
Per default tutti i comandi LPRINT ed LLIST inviano i caratteri all'interfaccia parallela; possono naturalmente essere usate anche stampanti seriali, dopo averle attivate per mezzo del comando DOS MODE, discusso nel Capitolo 3: questo comando è in grado di redirigere l'output diretto ad una stampante parallela verso una stampante seriale. Rileggete il Capitolo 3 se vi occorrono maggiori dettagli sul funzionamento di questo comando.
In questa parte del capitolo trattiamo del controllo della stampante dall'interno del BASIC. Vi sottolineiamo che alcuni comandi di stampa — primi fra tutti i caratteri di controllo — qui presentati non funzioneran-

no con qualsiasi tipo di stampante e perciò alcune funzioni saranno utili solo sulla stampante IBM 80 a matrice o sulle stampanti Epson MX.

## LE ISTRUZIONI LPRINT E LPRINT USING

Per inviare dati alla stampante potete usare i due comandi LPRINT o LPRINT USING, la cui sintassi è:

LPRINT [*lista di espressioni*] [;]
LPRINT USING X$; *lista di espressioni* [;]

Le due istruzioni hanno lo stesso comportamento di PRINT e PRINT USING; se un punto e virgola o uno spazio separa i termini della lista, ogni termine viene stampato con l'aggiunta di uno spazio bianco alla fine e, in caso di numeri positivi, con anche uno spazio bianco che lo precede. Se invece i termini sono separati per mezzo di virgole, vengono stampati all'inizio delle varie zone di stampa, come descritto nel Capitolo 5. Se nessuna espressione segue la parola LPRINT, verrà stampata una riga bianca. Il programma:

```
10 X=45
20 LPRINT 20.4444;99;1;X
30 LPRINT
40 LPRINT
50 LPRINT 20.4444,99,1,X
```

avrà perciò come output:

```
20.4444  99  1  45

20.4444        99             1              45
```

Se un'istruzione LPRINT o LPRINT USING è seguita da un punto e virgola, l'istruzione successiva dello stesso tipo continuerà a scrivere sulla stessa riga finché questa non termina, proprio come accade per le istruzioni PRINT e PRINT USING. Inoltre, i caratteri di formato che compaiono nella maschera X$ sono gli stessi dell'istruzione PRINT USING. Le seguenti istruzioni:

```
5 X = 45
10 LPRINT
20 LPRINT "UN OUTPUT";
```

```
30 LPRINT " SULLA ";
40 LPRINT "STESSA RIGA,SEGUITO DA";
50 LPRINT " DATI FORMATTATI"
60 LPRINT USING "##.##    ";20.444,99,1,X
```

produrranno in stampa questo output:

```
UN OUTPUT SULLA STESSA RIGA,SEGUITO DA DATI FORMATTATI
20.44    99.00    1.00    45.00
```

**Larghezza di una riga di stampa**

L'istruzione LPRINT assume per default che la larghezza della riga di stampa sia di 80 caratteri: cioè verrà eseguita un'operazione di a capo ogni 80 caratteri stampati sulla stessa riga. La larghezza può però essere modificata per mezzo dell'istruzione WIDTH, in questo modo:

WIDTH "LPT1:",*n*

in cui il parametro *n* è un'espressione il cui valore può variare tra 0 e 255. Tenete conto del fatto che se l'istruzione LPRINT produce una riga lunga esattamente *n* caratteri (o 80, nel caso di default) il cursore sarà mandato a capo una volta in più.
Se specificate una larghezza di linea maggiore di 80 caratteri quando la vostra stampante è ad 80 caratteri, potete risolvere il problema sfruttando l'opzione di stampa compressa, ottenibile con il seguente comando:

```
LPRINT CHR$(15)
```

Per ritornare alla stampa normale inviate invece il comando:

```
LPRINT CHR$(18)
```

Specificando la massima larghezza possibile per la riga di stampa, cioè 255 caratteri, in pratica disabilitate il ritorno carrello e il ritorno a capo automatico delle istruzioni PRINT ed LPRINT.
Per mezzo del comando:

```
WIDTH "LPT1:",255
```

in realtà create una linea di larghezza "infinita". Sarà perciò compito del programma inviare opportuni comandi di *line feed* (salto di una riga) — comando CHR$(10) — e di ritorno carrello — comando CHR$(13) —. La li-

nea infinita risulta molto utile quando la stampante ha capacità grafiche speciali. A queste stampanti, infatti, è possibile inviare righe lunghe parecchie centinaia di caratteri.

## CODICI DI CONTROLLO DELLA STAMPANTE

I codici di controllo consistono in sequenze di caratteri non stampabili ed a volte di caratteri standard con un preciso significato: potete inviare i codici alla stampante per mezzo dell'istruzione LPRINT e della funzione CHR$. Nel paragrafo precedente abbiamo usato il carattere di controllo 15 per inviare alla stampante il segnale di stampa compressa.
La Tabella 6.3 contiene i principali codici di controllo riconosciuti dalle stampanti IBM, Epson MX-80 ed Epson MX-100.

### Come cambiare gli stili di stampa

Esistono cinque tipi fondamentali di stampa: normale, espansa, compressa, enfatizzata e ribattuta. Ognuno di questi stili può essere abilitato o disabilitato per mezzo di alcuni particolari codici di controllo; vi presentiamo un programma che mostra gli stili possibili ed i relativi codici:

```
10 LPRINT "Questa e' una stampa normale"
20 LPRINT CHR$(14) "Questa e' una stampa espansa." '
 "14" per stampa espansa
30 LPRINT CHR$(15) "Questa e' una stampa compressa."
 ' "15" per stampa compressa
40 LPRINT CHR$(18); ' Questo per cancellare la stampa
 compressa
50 LPRINT CHR$(27) "E"; ' "27" seguito dalla lettera
"E" per la stampa enfatizzata
60 LPRINT "Questa e' una stampa enfatizzata."
70 LPRINT CHR$(27) "F"; ' Questa sequenza per elimina
re la stampa enfatizzata
80 LPRINT CHR$(27) "G"; ' Questa sequenza per stampa
ribattuta
90 LPRINT "Questa e' una stampa ribattuta."
100 LPRINT CHR$(27) "H"; ' Questa sequenza per cancel
lare la stampa ribattuta
110 LPRINT "E questa e' di nuovo una stampa normale."
```

Eseguendo il programma si ottiene il seguente output:

```
Questa e' una stampa normale
Questa e' una stampa espansa.
Questa e' una stampa compressa.
Questa e' una stampa enfatizzata.
Questa e' una stampa ribattuta.
E questa e' di nuovo una stampa normale.
```

**Tabella 6.3** Caratteri di controllo della stampante

| Azione | Sequenza di caratteri | Commenti |
|---|---|---|
| Stampa espansa | CHR$(14) | Automaticamente eliminata alla fine della stampa della riga |
| Cancella stampa espansa | CHR$(4) | Per cancellarla nel mezzo di una riga |
| Stampa compressa | CHR$(15) | Influisce su tutti i dati che compaiono nella stessa linea del comando CHR$(15) |
| Cancella stampa compressa | CHR$(18) | |
| Stampa enfatizzata | CHR$(27) "E" | La velocità di stampa diventa 40 caratteri per secondo |
| Cancella stampa enfatizzata | CHR$(27) "F" | |
| Stampa ribattuta | CHR$(27) "G" | |
| Cancella stampa ribattuta | CHR$(27) "H" | |
| Spaziatura tra righe 1/6″ | CHR$(27) "2" | Spaziatura di default |
| Spaziatura tra righe 1/8″ | CHR$(27) "0" | |
| Spaziatura tra righe 7/72″ | CHR$(27) "1" | |
| Predispone le tabulazioni verticali | CHR$(27) "B" *n*1 *n*2... *n*K CHR$(0) | *n* può variare da 1 a 66<br>K può arrivare al massimo a 64 |
| Tabulazioni verticali | CHR$(11) | |
| Cambio pagina | CHR$(12) | |
| Campanello | CHR$(7) | |

È consentito anche mescolare in vari modi questi stili di stampa, per ottenerne di nuovi; la seguente istruzione produce caratteri espansi e ribattuti:

```
10 LPRINT CHR$(14) CHR$(27) "G" "QUESTO E' MOLTO IMPO
RTANTE"
```

Quando eseguite questa linea di programma viene stampata la riga:

```
QUESTO E' MOLTO IMPORTANTE
```

**Comandi di movimento della stampante**

La stampante è in grado di effettuare due movimenti: verticale, per far avanzare la carta, oppure orizzontale, muovendo la testina di stampa da sinistra a destra. Anche questi movimenti possono essere controllati per mezzo di alcuni comandi.
La spaziatura tra le righe è il risultato del movimento verticale: lo spazio lasciato tra due righe consecutive per default è fissato a 1/6 di pollice, ma può essere posto uguale a 1/8 o a 7/72 di pollice con dei comandi di controllo della stampante. Ecco un programma che esemplifica questa azione:

```
10 GOSUB 100
20 LPRINT CHR$(27) "0"
30 GOSUB 100
40 LPRINT CHR$(27) "1"
50 GOSUB 100
55 LPRINT CHR$(27) "2"
60 END
100 LPRINT "Potete scrivere su di una pagina"
110 LPRINT "diverse quantita' di testo"
120 LPRINT "variando la spaziatura tra due"
130 LPRINT "linee successive."
140 LPRINT " Ora stiamo usando spaziature di"
150 LPRINT "1/6, 1/8 e 7/72 di pollice tra due linee."
160 LPRINT
170 RETURN
```

Un simile programma produce un output di questo tipo:

```
Potete scrivere su di una pagina
diverse quantita' di testo
variando la spaziatura tra due
linee successive.
```

```
  Ora stiamo usando spaziature di
1/6, 1/8 e 7/72 di pollice tra due linee.


Potete scrivere su di una pagina
diverse quantita' di testo
variando la spaziatura tra due
linee successive.
  Ora stiamo usando spaziature di
1/6, 1/8 e 7/72 di pollice tra due linee.

Potete scrivere su di una pagina
diverse quantita' di testo
variando la spaziatura tra due
linee successive.
  Ora stiamo usando spaziature di
1/6, 1/8 e 7/72 di pollice tra due linee.
```

La prima volta in cui viene stampato il testo, è utilizzata la spaziatura di default, cioè 1/6 di pollice tra una riga e l'altra; nella linea 20 vengono inviati alla stampante, per mezzo di un'istruzione LPRINT, i caratteri di controllo CHR$(27) e 0, in modo da definire una spaziatura di 1/8 di pollice; con la stessa procedura, nella linea 40 viene inviato alla stampante il carattere CHR$(27) seguito da 1, in modo da porre la spaziatura tra le righe uguale a 7/72 di pollice. Per ritornare al modo normale, cioè 1/6 di pollice, è stata inserita l'istruzione 55, che invia alla stampante i caratteri CHR$(27) e 2.

Ecco ora un programma che mostra come spaziatura e stili di stampa possano essere combinati o variati contemporaneamente:

```
10 LPRINT CHR$(14) "Potete stampare una piccola "
20 LPRINT CHR$(14) "quantita' di caratteri"
30 LPRINT CHR$(14) "molto grandi per evidenziare"
40 LPRINT CHR$(14) "un testo o renderlo facile"
50 LPRINT CHR$(14) "da leggere, ad esempio per"
60 LPRINT CHR$(14) "persone dalla vista debole."
70 LPRINT CHR$(27) "1" CHR$(15)
80 LPRINT  " D'altra parte potete anche usare  uno
            stile di stampa compresso con"
90 LPRINT  "una spaziatura piccola per inserire
            il maggior numero di informazioni"
100 LPRINT "possibile in un'area di pagina definita."
110 LPRINT CHR$(27) "2"; CHR$(18)
```

L'esecuzione di questo programma produce l'output seguente:

```
Potete stampare una piccola
quantita' di caratteri
molto grandi per evidenziare
un testo o renderlo facile
```

```
da leggere, ad esempio per
persone dalla vista debole.

 D'altra parte potete anche usare  uno
stile di stampa compresso con
una spaziatura piccola per inserire
il maggior numero di informazioni
possibile in un'area di pagina definita.
```

Osservate come, per stampare caratteri espansi, dovete inviare il comando CHR$(14) prima di ogni riga: infatti, come già ricordato in Tabella 6.3, la stampa espansa viene disabilitata alla fine di ogni riga stampata; osservate anche che la combinazione di comandi in linea 70 fa sì che la stampante usi caratteri compressi con una spaziatura di 7/72 di pollice. La linea 110 serve per riportare la stampa alle condizioni normali.
Le tabulazioni per la stampa possono essere impostate sia in direzione orizzontale che in direzione verticale; per ottenere la tabulazione verticale dovete innanzitutto definire la posizione di tabulazione con l'istruzione:

LPRINT CHR$(27) "B" *n*1 "H" *n*2 "H" ... *n*K "H" CHR$(0);

Le posizioni di tabulazione sono specificate dal parametro *n*K, in cui *n* può variare da 1 a 66, ed il numero massimo di posizioni è 64. Ogni posizione, espressa in esadecimale, deve essere seguita dalla lettera H (maiuscola) ed il carattere "nullo" — cioè CHR$(0) — deve concludere la sequenza.
Ecco l'istruzione che permette di predisporre tre tabulazioni verticali, alle righe 16, 21 e 52:

```
LPRINT CHR$(27) "B" 10 "H" 15 "H" 34 "H" CHR$(0);
```

Per far avanzare la stampante dalla posizione in cui si trova alla successiva posizione di tabulazione verticale, dovete inviare il seguente comando:

```
LPRINT CHR$(11);
```

Potete anche far avanzare la stampante di una pagina intera alla volta, per mezzo del comando:

```
LPRINT CHR$(12);
```

Per tabulazioni orizzontali usate la funzione TAB(*n*), esattamente come per le istruzioni PRINT. In questo caso, *n* può variare tra 1 e la larghezza definita per la stampante. Se TAB(*n*) indica una posizione che precede quella occupata dal cursore, la stampante avanza fino alla riga successiva.
A titolo di esempio, costruiamo l'intestazione per una tabella che contiene quattro colonne di dati; la seguente istruzione:

```
10 LPRINT "- Nome cliente -"TAB(21)"- Somma dovuta -"
TAB(39)"- Valore ordine -"TAB(59)"- Saldo -"
```

stampa l'intestazione:

```
- Nome cliente -     - Somma dovuta -  - Valore ordine -    - Saldo -
```

La funzione TAB(*n*) non può però essere utilizzata all'interno dell'istruzione LPRINT USING: se volete ottenere una tabulazione verticale in questo caso dovete inserire il numero opportuno di spazi nella maschera. Ecco il programma completo che stampa la tabella:

```
10 LPRINT "- Nome cliente -"TAB(21)"- Somma dovuta -"
TAB(39)"- Valore ordine -"TAB(59)"- Saldo -"
20 LPRINT
30 FOR I=1 TO 4
40 READ CLIENTE$(I),SOMMA(I),ORDINE(I),SALDO(I)
50 LPRINT  USING "\                \";CLIENTE$(I);
60 LPRINT  USING "      ######       ";SOMMA(I),ORDINE(
I),SALDO(I)
70 NEXT I
80 DATA "ROSSI, MARIO", 42000, 238000, 280000,"COLOMB
O, MARTA", 1000, 0, 1000,"LORENZI, GIANFRANCESCOMARIA
", 45000, 22000, 67000, "MANOLI, BEPPE", 620000, 8000
, 628000
```

Dall'esecuzione del programma si ottiene:

```
- Nome cliente -     - Somma dovuta -  - Valore ordine -    - Saldo -

ROSSI, MARIO                 42000            238000          280000
COLOMBO, MARTA                1000                 0            1000
LORENZI, GIANFRANCES         45000             22000           67000
MANOLI, BEPPE               620000              8000          628000
```

## 6.3 Tecniche di input da tastiera

Nel Capitolo 5 abbiamo introdotto le più semplici tecniche di utilizzo della tastiera come dispositivo per generare input; in questo paragrafo miglioreremo queste tecniche allo scopo di creare programmi più facili da usare.

## CONSIDERAZIONI SU COME PROGRAMMARE L'INPUT

Sono due i punti principali da considerare quando si richiede all'operatore di inserire dati in un programma: innanzitutto l'input dev'essere controllato in modo da evitare errori; in secondo luogo, l'operatore dev'essere in grado di capire esattamente quale informazione sia necessaria in ogni momento.
Idealmente, infatti, un programma non dovrebbe risentire di un input sbagliato: voi non dovreste essere in grado di far terminare il programma o di ottenere comportamenti imprevisti inserendo informazioni scorrette.
Il controllo sull'input dovrebbe essere effettuato immediatamente dopo che il dato è stato inviato al programma: se questo non è corretto (al di fuori dei valori richiesti dal programma, valori senza significato, e così via..) l'operatore dovrebbe esserne informato e il processo di input ripetuto.
L'istruzione INPUT esegue già alcuni di questi controlli: in particolare, il BASIC visualizza il messaggio ?REDO FROM START (Ripeti dall'inizio) se il tipo del dato in input non corrisponde a quello della variabile a cui va assegnato. Dopo il messaggio, il BASIC resta in attesa di un nuovo dato dall'operatore. Questo tipo di errore può essere provocato dal tentativo di inviare caratteri alfabetici quando la variabile dell'istruzione INPUT è numerica o se il numero di termini inseriti non corrisponde al numero delle variabili presenti.
Inoltre, se il numero inserito non ha la stessa precisione della variabile a cui dev'essere assegnato, il processo non sarà completo e verrà visualizzato un messaggio d'errore; se rispondete 32999 al prompt visualizzato dall'istruzione:

```
INPUT "Inserire il numero di angeli che stanno su di
una capocchia di spillo" ; R%
```

otterrete il messaggio d'errore OVERFLOW.
Il modo più semplice per rendere utilizzabile il programma senza grossi problemi è di visualizzare delle direttive molto chiare ogni volta che il programma richiede un input da parte dell'operatore; ricorderete, naturalmente, che il comando INPUT permette di aggiungere una stringa che viene visualizzata prima del prompt. Una stringa ben congegnata avvisa l'operatore di che tipo di dato sia richiesto, in che formato e, se necessario, in quale intervallo di valori. Un esempio:

```
110 INPUT "Inserite il vostro nome, seguito da una vi
rgola e la vostra abilita'(da 1 a 10) --" , XNOME$ ,
LIVELLO%

Inserite il vostro nome, seguito da una virgola e la
vostra abilita'(da 1 a 10)--Franco,7
```

Se il programma richiede all'operatore di scegliere fra numerose possibilità, può essere utile un approccio detto "a menu". Un menu (cioè una lista di opzioni) presenta le scelte possibili, ordinate e numerate. Un menu potrebbe apparire così:

```
    *** SELEZIONE DEL CALCOLO ***

Questo programma utilizzerà i parametri che voi
avete scelto per calcolare una delle seguenti
quantità:
   1. La distanza del corpo celeste
   2. Il periodo di rivoluzione
   3. La velocità di rotazione

Fate la vostra scelta battendo il numero
corrispondente (1,2,3); se volete ritornare
al menu principale battete "4".
```

In questo caso, titolo, descrizione e opzioni sono generate per mezzo di una serie di istruzioni PRINT. L'ultima linea, invece, "Fate la vostra scelta..." è la stringa-prompt dell'istruzione INPUT.

## L'ISTRUZIONE LINE INPUT

L'istruzione LINE INPUT ha un comportamento molto simile all'istruzione INPUT che abbiamo finora utilizzato, poiché riceve input dalla tastiera e li assegna ad una variabile predefinita. La sua sintassi è la seguente:

LINE INPUT [;] ["*stringa prompt*";] *varstr*

e può accogliere una stringa lunga fino a 254 caratteri ed assegnarla alla variabile di tipo stringa indicata dal parametro *varstr*. Se battete più di 254 caratteri in risposta al prompt, quelli in eccesso verranno troncati e perciò andranno perduti.
La differenza principale tra le due istruzioni INPUT e LINE INPUT è che con la seconda tutti i caratteri in input vengono letti ed assegnati alla variabile: una virgola battuta sulla tastiera entrerà così a far parte della variabile *varstr* come virgola e non come delimitatore tra due diversi termini, come accadrebbe con l'istruzione INPUT.
Se aggiungete il punto e virgola dopo la parola INPUT, non verrà eseguito un ritorno a capo: ciò significa che il cursore resterà al termine della riga appena scritta invece di spostarsi alla riga seguente.
Questa istruzione è utile quando vi interessa inserire del testo in un pro-

gramma, come nel seguente, in cui alla linea 120 un'istruzione LINE INPUT è in attesa di un breve commento che riguarda uno specifico paziente e lo memorizza nell'array di tipo stringa NOTA$(I); il testo ora fa parte di un'unica registrazione, propria di quel paziente. Più avanti, nella linea 580, la storia del paziente viene visualizzata, esattamente come era stata scritta.

```
      .
      .
      .
100 FOR I = 1 TO 100
110 INPUT "INSERISCI IL NOME DEL PAZIENTE:",PNOME$(I)
120 LINE INPUT "INSERISCI UN BREVE COMMENTO (NON PIU'
LUNGO DI 254 LETTERE O CIRCA 4 RIGHE DI SCHERMO)"; NO
TA$(I)
130 NEXT I
      .
      .
      .
500 FOR I = 1 TO 100
510 LOCATE 1,25
520 PRINT "IL NOME DEL PAZIENTE E' - "  PNOME$(I) SPC
(10)
530 FOR X = 0 TO 5 ' CANCELLA UN'AREA DI SCHERMO PER
LA NOTA
540 LOCATE 10+X,1
550 PRINT SPC(79)
560 NEXT X
570 LOCATE 10,5 ' RIENTRO DELL'INIZIO DELLA NOTA
580 PRINT NOTA$(I)
590 LOCATE 20,10
600 PRINT "PREMI UN TASTO QUALSIASI PER PASSARE AL PA
ZIENTE SUCCESSIVO"
610 T$=INKEY$:IF T$="" THEN 610
620 LOCATE 20,1:PRINT SPC(80) ' CANCELLA IL MESSAGGIO
630 NEXT I
      .
      .
      .
```

Avrete notato anche altre particolarità di questo programma: le linee dalla 100 alla 130 sono predisposte per creare 100 schede di pazienti; le linee dalla 500 alla 630 visualizzano le schede memorizzate, una per volta; nella linea 520 la funzione SPC gioca un ruolo fondamentale: assicura che tutti i caratteri rimanenti non coperti dal nuovo nome siano cancellati per non creare confusione; nello stesso modo, il loop che va dalla linea 530 alla 560 cancella lo spazio destinato a contenere la nota successiva. Esaminate poi attentamente le linee 600 e 610: queste mostrano un ulteriore mezzo di comunicazione tra tastiera e programma; abbiamo infatti

creato un loop che provoca una pausa nel programma per permettere all'operatore di leggere con comodo il testo visualizzato.

## INPUT PER MEZZO DEI TASTI FUNZIONE

Nel Capitolo 4 abbiamo visto come i tasti funzione possano essere usati in modo diretto per comunicare con il BASIC: premendo uno di questi tasti, infatti, inviamo una specifica stringa di caratteri al PC. Ora vi mostriamo come i tasti funzione possano essere utilizzati anche per rispondere ad un'istruzione INPUT o LINE INPUT.
Il BASIC, quando viene avviato, possiede un insieme di stringhe predefinite, una per ogni tasto funzione: voi avete la possibilità di ridefinire queste stringhe all'interno di un programma con l'ausilio dell'istruzione KEY, la cui sintassi è:

KEY *n*,X$

dove *n* è un numero che va da 1 a 10 ed indica a quale tasto funzione fate riferimento e la variabile di tipo stringa X$, lunga al massimo 15 caratteri, rappresenta la stringa che volete assegnare a quel particolare tasto. Se volete ottenere un tasto funzione che esegua anche l'equivalente del tasto ENTER, concatenate la funzione CHR$(13) (ritorno carrello) alla stringa X$.
Per visualizzare l'elenco delle assegnazioni dei tasti, usate l'istruzione:

```
KEY LIST
```

che produrrà una lista di tutte le stringhe assegnate ai dieci tasti funzione. Le due istruzioni KEY OFF e KEY ON vi permettono rispettivamente di disabilitare o abilitare la visualizzazione dei tasti funzione nella venticinquesima riga dello schermo. Le istruzioni vengono eseguite in questa semplice forma:

```
KEY ON
KEY OFF
```

Se lo schermo è predisposto per 40 colonne, nella riga 25 compaiono, in seguito all'istruzione KEY ON, solo i primi cinque tasti, mentre uno schermo di 80 colonne permetterà la visualizzazione di tutti e dieci.
Se state usando la funzione INKEY$ in Cassette BASIC per leggere le stringhe assegnate ai tasti funzione, dovreste eseguire un'istruzione DEF SEG seguita da un POKE 106,0 dopo aver ricevuto l'ultimo carattere della stringa. L'istruzione:

```
DEF SEG : POKE 106,0
```

infatti cancella il buffer di tastiera in modo da evitare di generare errori nell'input. Potreste anche utilizzare questa istruzione prima di un loop di pausa in cui viene controllata la tastiera (ad esempio, le linee 600 e 610 dell'esempio precedente) in modo da assicurarvi che il loop sia interrotto solo da un'azione specifica sulla tastiera.
Ecco un programma che vi mostra come usare i tasti funzione per l'input:

```
        .
        .
        .
140 KEY 1,"RIPETE"+CHR$(13)
150 KEY 2,"PASSO"+CHR$(13)
160 KEY 3,"INCREMENTO"+CHR$(13)
170 KEY 4,"DECREMENTO"+CHR$(13)
        .
        .
        .
230 KEY 10,"FINE"+CHR$(13)
240 DEF SEG : POKE 106,0
        .
        .
        .
1100 INPUT "Vuoi vedere i tasti funzione (Y/N)";G$
1110 IF G$<>"Y" THEN 1180
1120 ORIZ=POS(0):VERT=CSRLIN:SCREEN 0,1,2,2
1130 CLS:KEY OFF
1140 KEY LIST
1150 LOCATE 25,1:PRINT "Premi un tasto qualsiasi per
continuare"
1160 X$=INKEY$:IF X$="" THEN 1160
1170 SCREEN 0,1,0,0:LOCATE VERT,ORIZ
1180 .
        .
        .
```

Le linee dalla 140 alla 230 di questo programma servono per l'assegnamento delle stringhe ai tasti funzione; le linee 1100 e 1110 richiedono all'operatore il comando per la visualizzazione dell'elenco delle stringhe assegnate: in caso affermativo, la posizione corrente del cursore viene memorizzata nelle variabili ORIZ e VERT (assumendo che la pagina corrente di output sia la 0). Infine, la linea 1120 prepara la pagina 2 dello schermo (sia l'area normale per i tasti che la venticinquesima riga), la 1130 la cancella e visualizza le stringhe assegnate ai tasti funzione. Le li-

nee 1150 e 1160 costituiscono un loop che permette all'operatore di leggere l'output. Per concludere, la linea 1170 ripristina lo schermo iniziale e riporta il cursore nella posizione originale.

## 6.4 Intercettamento di eventi

Una procedura detta intercettamento di eventi (*event trapping*), disponibile solo con l'Advanced BASIC, permette di interrompere il programma in un punto qualsiasi mediante la lettura di diversi dispositivi: la tastiera, una penna ottica, un joystick, un contatore, oppure un dispositivo esterno possono in questo caso far scattare l'intercettamento.
Potreste creare una possibilità di interrompere il programma inserendo istruzioni GOSUB che ad intervalli regolari facciano saltare ad una routine di input, ma questo richiederebbe una inutile quantità di salti. L'intercettamento di eventi consente di gestire l'interruzione senza aggiunte che renderebbero il programma più lento.

### COME USARE L'INTERCETTAMENTO DI EVENTI

Per usare questa procedura, dovete innanzitutto abilitare esplicitamente un evento a funzionare da "evento di interruzione" e poi indicare la locazione in cui si trova una routine di intercettamento, il tutto per mezzo delle seguenti istruzioni:

*evento* ON
ON *evento* GOSUB *linea*

Il parametro *evento* può essere KEY(*n*) (per i tasti funzione e controllo cursore), STRIG(*n*) (per il joystick), PEN(*n*) (per la penna ottica), TIMER(*n*) (per controllare intervalli di tempo), PLAY(*n*) (per un accompagnamento musicale) o COM(*n*) (per un'interfaccia di comunicazione); il parametro *linea* dev'essere il numero di una linea del programma.
Quando l'istruzione *evento* ON è stata eseguita, il BASIC controlla, dopo l'esecuzione di ogni successiva linea di programma, se è accaduto l'evento indicato: in caso affermativo, l'esecuzione viene "intercettata", cioè viene temporaneamente sospesa l'esecuzione della linea successiva; il BASIC salta quindi alla linea indicata nell'istruzione ON *evento* GOSUB *linea*.

Supponiamo di voler interrompere un programma premendo il tasto funzione F6: le linee 50 e 55 dell'esempio predispongono questa possibilità:

```
      .
      .
      .
50 ON KEY(6) GOSUB 4000
55 KEY(6) ON
      .
      .
      .
1340 F=8.44E12*BETA
1350 G=BETA/ZIRCONIO
      .
      .
      .
3000 GOTO 1340
4000 ' Routine di intercettamento dovuta al tasto F6
      .
      .
      .
4500 RETURN
      .
      .
      .
```

Supponiamo che il tasto F6 venga battuto dopo l'esecuzione della linea 1340: l'esecuzione salta alla routine specificata nell'istruzione ON..GOSUB, in questo caso alla linea 4000. Dopo l'esecuzione dell'istruzione RETURN alla fine della routine di intercettamento, il BASIC ritorna alla linea che segue quella in cui è accaduto l'evento, in questo caso la 1350.
Un evento, perciò, è in grado di sospendere l'esecuzione del programma corrente e di richiamare l'esecuzione di una routine; quando un evento è abilitato (ON) ha la precedenza assoluta su quanto sia in esecuzione in quel momento.
L'evento che provoca la chiamata di una subroutine di intercettamento viene automaticamente disabilitato mentre la routine è in esecuzione: questo per evitare che un qualsiasi evento possa interrompere la propria routine di intercettamento. Di solito, poi, l'evento è riabilitato al termine della routine, cioè quando viene eseguita l'istruzione RETURN.

## EVENTI E ROUTINE DI INTERCETTAMENTO DI EVENTI

Discuteremo ora dei tipi di eventi suddividendoli in base al tipo di dispositivo che provoca l'intercettamento: la tastiera, una penna ottica, un joy-

stick, un intervallo di tempo, il buffer di accompagnamento musicale, un'interfaccia di comunicazione.

### Tastiera

L'Advanced BASIC è in grado di riconoscere l'uso di uno qualsiasi dei tasti funzione, dei tasti di controllo cursore (quelli posti nel tastierino numerico) e di altri sei tasti definiti in precedenza come eventi. Per abilitare questi eventi, potete usare l'istruzione KEY(*n*), dove *n* si riferisce ad uno dei 20 possibili tasti "generatori di eventi" elencati nella Tabella 6.4.

**Tabella 6.4** Tasti generatori di eventi

| Tasto | Numero del tasto |
|---|---|
| Tasti funzione F1-F10 | da 1 a 10 |
| Cursore in alto | 11 |
| Cursore a sinistra | 12 |
| Cursore a destra | 13 |
| Cursore in basso | 14 |
| Tasti predefiniti | da 15 a 20 |

Dopo l'esecuzione di un'istruzione KEY(*n*) ON, il premere uno qualsiasi dei tasti indicati farà saltare il programma alla linea specificata nella corrispondente istruzione ON KEY(*n*) GOSUB *linea*.
Ecco come questo intercettamento di eventi dovuti alla tastiera può essere inserito in un programma:

```
      .
      .
      .
110  ' Abilita il tasto funzione F1 ed i tasti di con
trollo del cursore
120 KEY(1) ON:KEY(11) ON:KEY(12) ON: KEY(13) ON:KEY(1
4) ON
130 ' Il tasto F1 fara' terminare il programma
140 ON KEY(1) GOSUB 10000
      .
      .
      .
4100 ' Esegue l'appropriata routine di movimento dell
a figura, a seconda del tasto premuto
4110 ON KEY(11) GOSUB 11500
4120 ON KEY(12) GOSUB 12000
4130 ON KEY(13) GOSUB 12500
4140 ON KEY(14) GOSUB 13000
```

```
      .
      .
      .
10000 END
      .
      .
      .
11500 ' Routine di movimento verso l'alto
      .
      .
      .
12000 ' Routine di movimento verso sinistra
      .
      .
      .
12500 ' Routine di movimento verso destra
      .
      .
      .
13000 ' Routine di movimento verso il basso
      .
      .
      .
```

La linea 120 abilita il tasto funzione F1 ed i tasti di controllo cursore; le routine di intercettamento sono indicate nella linea 140 e nelle linee che vanno dalla 4110 alla 4140.

Altre istruzioni dell'Advanced BASIC che controllano l'intercettamento dovuto ad un tasto premuto sono le istruzioni KEY(*n*) OFF e KEY(*n*) STOP; KEY(*n*) OFF ha la funzione opposta di KEY(*n*) ON: impedisce che un certo evento possa chiamare una routine di intercettamento finché non venga data disposizione contraria (esecuzione di un'istruzione KEY(*n*) ON). L'istruzione KEY(*n*) OFF è equivalente a ON KEY(*n*) GOSUB 0.

Anche l'istruzione KEY(*n*) STOP, pur con una sostanziale differenza, serve per disabilitare il tasto indicato: in seguito a questa istruzione, infatti, il BASIC continua a controllare se l'evento si compie o no. Se il tasto viene premuto, sul momento non accade nulla, ma non appena viene eseguita un'istruzione KEY(*n*) ON anche il tasto premuto precedentemente fa partire la routine di intercettamento.

L'istruzione KEY(*n*) STOP è utile in particolare quando il programma esegue una parte critica dal punto di vista temporale. L'istruzione KEY(*n*) STOP impedisce che un evento proveniente dalla tastiera interferisca con la parte di programma dipendente dal tempo, senza peraltro eliminare completamente la possibilità di utilizzare i tasti nel modo appena visto. Il seguente programma illustra questo metodo di disabilitazione temporanea degli eventi da tastiera:

```
      .
      .
      .
10 ' Il tasto F5 visualizza data e ora
20 KEY(5) ON
30 ON KEY(5) GOSUB 1000
      .
      .
      .
200 ' Routine in tempo reale
210 KEY(5) STOP
      .
      .
      .
300 ' Fine della routine; riabilita la visualizzazion
e di data e ora
310 KEY(5) ON
      .
      .
      .
500 GOTO 100
1000 ORIZ=CSRLIN:VERT=POS(0)  'Memorizza la posizione
del cursore
1010 SCREEN 0,1,2,2 ' Scrive data e ora su una pagina
di schermo non utilizzata
1020 CLS
1030 FOR I=0 TO 100
1040 LOCATE 10,20
1050 PRINT "Sono le ore " TIME$ " del giorno " DATE$
1060 NEXT I
1070 SCREEN 0,1,0,0 ' Ritorna alla pagina originale
1080 LOCATE ORIZ,VERT ' Ripristina il cursore
1090 RETURN
      .
      .
      .
```

Nel programma è mostrato anche un modo dinamico per visualizzare data e ora — vedi le linee che vanno dalla 1000 alla 1090.
Un ulteriore esempio di intercettamento di eventi dovuti alla tastiera è presentato in questo programma, in cui si assume che voi abbiate a disposizione una scheda Colore/Grafica; se non la possedete, eseguitelo senza le linee 90, 100, 180 e 240:

```
10 SCREEN 0,1,0,0:KEY OFF:CLS 'Inizia con la pagina 0
dello schermo pulita
20 GOSUB 260 'Visualizza i valori dei tasti funzione
30 KEY(1) ON:KEY(2) ON:KEY(10) ON ' Abilita i tasti
40 ON KEY(1) GOSUB 120
```

```
50 ON KEY(2) GOSUB 170
60 ON KEY(10) GOSUB 230
70 X=(X+1) MOD 30 ' Genera un ciclo da 0 a 30
80 PRINT SPC(X) "*^*^*^*^*^*^*^*^*^*^"
90 Y=(Y+1) MOD 16 ' Genera tutti i colori
100 COLOR Y
110 GOTO 70 ' Riempie lo schermo
120 CLS
130 LOCATE 10,30
140 PRINT "BENE, ORA E' PULITO":FOR I = 0 TO 1000:
    NEXT I ' Visualizza il messaggio--Ritardo
150 GOSUB 260 ' Visualizza il valore dei tasti funzio
ne
160 RETURN
170 SCREEN 0,1,2,2:CLS ' Usa un'altra pagina di scher
mo
180 COLOR 27
190 LOCATE 14,40 ' Localizza il messaggio
200 PRINT "EEEEEEEEEEEEEEK!!!!!!"
210 FOR I = 0 TO 2000:NEXT I ' Ritardo
220 SCREEN 0,1,0,0:RETURN ' Torna allo schermo inizia
le
230 CLS
240 COLOR 7
250 KEY ON:END 'Ritorna al punto di partenza, qualunq
ue fosse
260 LOCATE 25,1:PRINT "F1 VALE CLEAR, F2 VALE ANGST,
F10 VALE END" 'Visualizza i tasti funzione sull'ultim
a riga di schermo
270 RETURN
```

Osservate come la linea 20 chiama una routine che visualizza nella venticinquesima riga dello schermo l'azione dei tasti funzione abilitati: questo è un procedimento necessario in quanto le linee di programma dalla 70 alla 110 generano un disegno che riempie continuamente lo schermo; notate anche che la routine in linea 170 usa una diversa pagina di schermo al fine di preservare lo schermo principale.

Oltre agli eventi finora ricordati, come i tasti funzione e i tasti di controllo del cursore, l'istruzione KEY permette di abilitare altri sei eventi definibili dall'utente: un qualsiasi tasto può essere usato, in unione con i tasti CTRL, SHIFT, CAPS LOCK e NUM LOCK per produrre un evento di intercettamento. La sintassi per definire uno di questi eventi è la seguente:

KEY *n*,CHR$(*stato-shift*)+CHR$(*codice di scansione*)

La variabile *n* può assumere valori compresi tra 15 e 20 ed indica quale tra i sei eventi si definisce; la variabile *stato-shift* determina lo stato dei tasti speciali, con questi possibili valori:

**Valori di stato-shift**

| Tasto | Hex | Binario | Decimale |
|---|---|---|---|
| Non shiftato | &H0 | 0000 0000 | 0 |
| SHIFT | &H01 | 0000 0001 | 1 |
| CTRL | &H04 | 0000 0100 | 4 |
| ALT | &H08 | 0000 1000 | 8 |
| NUM LOCK | &H20 | 0010 0000 | 32 |
| CAPS LOCK | &H40 | 0100 0000 | 64 |

I due tasti SHIFT sono quelli che permettono di creare una condizione di shift: un tasto che per definizione debba usare uno di questi due tasti SHIFT richiederà indifferentemente l'uno dei due. Come potete constatare osservando il valore binario di questi tasti, i vari stati di shift sono stati pensati in modo da non interferire gli uni con gli altri quando vengono combinati.
I due tasti CTRL e ALT potrebbero così essere combinati per produrre uno stato di shift del valore &H0C (00001100 in binario): per creare un evento da intercettare, potrebbe ad esempio essere necessario premere contemporaneamente i tasti CTRL e ALT insieme ad un altro tasto a scelta. I tasti non shiftati possono essere definiti con uno stato di shift pari a 0.
Il parametro *codice di scansione* nell'istruzione KEY può assumere valori compresi tra 1 e 83 e serve per identificare il tasto che, in combinazione con i tasti di shift, produce l'evento richiesto. Nella Figura 6.3 è riportata la disposizione dei tasti nella tastiera del PC con il *codice di scansione* assegnato a ciascuno di essi, cioè il numero che identifica ciascuno degli 83 tasti del PC.

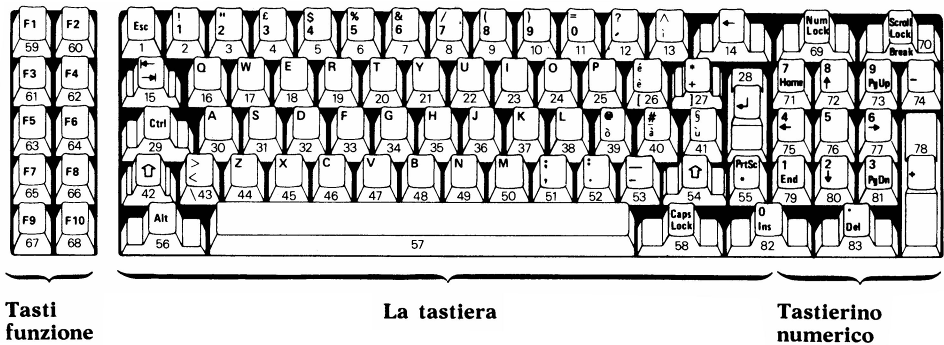



**Figura 6.3** La tastiera del PC-IBM, in cui compare il codice di scansione per ogni tasto

Dopo aver definito un evento per mezzo dell'istruzione KEY potete usare le istruzioni KEY(*n*) (con ON, OFF o STOP) anche per i valori di *n* che vanno da 15 a 20; nello stesso modo, il comando ON KEY(*n*) può essere usato per richiamare la routine di intercettamento per la relativa combinazione (sempre nell'intervallo di valori *n*, cioè fino a 20, come definito nell'istruzione KEY).
Supponiamo che un programma debba intraprendere una particolare azione ogni volta che vengono premuti i tasti ALT e Q (il tasto ALT deve restare premuto mentre si batte il tasto Q): dalla Figura 6.3 ricaviamo che il *codice di scansione* di Q è 16; il valore di shift del tasto ALT è &H08. Nel programma, perciò, devono comparire le seguenti linee:

```
10 KEY 15,CHR$(&H08)+CHR$(16) 'Assegna la sequenza
ALT-Q all'evento 15
20 KEY (15) ON
30 ON KEY (15) GOSUB 2000
      .
      .
      .
      .
      .
2000 'La combinazione ALT-Q provoca l'evento
      .
      .
2100 RETURN 'Ritorno dalla routine chiamata dall'even
to
```

**Penna ottica**

Una penna ottica può essere collegata al PC-IBM per mezzo della scheda Colore/Grafica: viene utilizzata posizionandone la punta nel punto dello schermo richiesto e premendo un interruttore in modo da "leggere" le coordinate del punto scelto.
L'Advanced BASIC supporta l'uso della penna ottica come intercettatore di eventi: questa può essere abilitata mediante l'istruzione PEN ON, che non ha argomenti, in quanto una sola penna è collegabile per mezzo della scheda Colore/Grafica.
Quando poi si accende l'interruttore per la penna ottica, l'Advanced BASIC fa riferimento all'ultima istruzione ON PEN GOSUB eseguita per ricavare il numero della linea a cui saltare e quindi esegue la routine di intercettamento relativa; un'istruzione ON PEN GOSUB 0 disabilita l'evento rilevato dalla penna ottica.
Le istruzioni PEN OFF e PEN STOP hanno lo stesso comportamento di KEY OFF e KEY STOP: PEN OFF disabilita l'evento, mentre PEN STOP

lo fa solo temporaneamente, ma mantiene il ricordo di ogni evento; una successiva istruzione PEN ON avrebbe come effetto immediato il salto alla routine di intercettamento se l'interruttore fosse stato premuto mentre era in vigore l'istruzione PEN STOP.
Per determinare le coordinate della penna ottica potete usare la funzione PEN (*n*): il parametro *n* varia da 0 a 9 e specifica quale tra i dieci valori possibili (le coordinate x e y sullo schermo, un valore che indica se la penna è accesa o spenta...) deve essere assegnato alla variabile. Fate riferimento all'Appendice A per l'elenco dei valori assunti dalla funzione PEN(*n*).
Ecco un programma che mostra l'uso della penna ottica:

```
      .
      .
      .
100 PEN ON
110 ON PEN GOSUB 1000
      .
      .
      .
900 GOTO 150
1000 COL=PEN(7):ROW=PEN(6) ' Posizione a cui la penna
sta puntando
1010 X=SCREEN(ROW,COL) ' Valore ASCII del carattere a
cui sta puntando la penna
1020 X$=CHR$(X) ' Converte il codice in carattere
1030 GOSUB 2000 ' Interpreta il carattere
1040 RETURN
      .
      .
      .
```

La penna ottica come intercettatore di eventi viene abilitata nella linea 100; quando la penna viene attivata, il programma salta alla routine di linea 1000, in cui la funzione PEN(*n*) è usata con argomenti 7 e 6 che danno, rispettivamente, la colonna e la riga relative alla posizione in cui la penna è stata attivata.

### Joystick e paddle

Anche il joystick ed il paddle possono essere usati per causare eventi in un programma scritto in Advanced BASIC: i pulsanti del joystick e del paddle sono controllati per mezzo delle istruzioni STRIG(*n*) ON, ON STRIG(*n*) GOSUB *linea*, STRIG(*n*) OFF e STRIG(*n*) STOP, che funzionano analogamente a quelle della penna, con l'accortezza di definire il valore

di *n*. L'argomento *n* è un'espressione numerica che vale 0 (per il pulsante A1), 2 (per il B1), 4 (per l'A2) o 6 (per il B2).
Il seguente programma mostra come utilizzare il joystick per rilevare eventi in un programma di gioco:

```
       .
       .
       .
100 STRIG(0) ON : STRIG(2) ON ' Abilita il joystick
       .
       .
       .
300 ' Parte il controllo della nave
310 ON STRIG(0) GOSUB 1000 ' Se viene premuto il puls
ante A1 spara bombe nucleari contro gli invasori
320 ON STRIG(2) GOSUB 2000 ' Se viene premuto il puls
ante B passa nell'iperspazio
330 GOSUB 500 ' Muove la nave in accordo con gli spos
tamenti del joystick
       .
       .
       .
900 GOTO 330
1000 STRIG(2) STOP ' Disabilita il pulsante B
       .
       .          ' Fuoco nucleare contro gli invasori
       .
1030 STRIG(2) ON ' Riabilita il pulsante B
1040 RETURN
       .
       .
       .
2000 STRIG(0) STOP ' Disabilita il pulsante A
       .
       .          ' Passaggio all'iperspazio
       .
2100 STRIG(0) ON ' Riabilita il pulsante A
2110 RETURN
       .
       .
       .
```

Le linee 100, 310 e 320 abilitano gli eventi creati dai pulsanti del joystick e definiscono le routine di intercettamento relative; ognuna di queste routine inizia con il disabilitare gli altri pulsanti del joystick per mezzo dell'istruzione STRIG(*n*) STOP (linee 1000 e 2000) in modo da evitare che le routine si interrompano a vicenda quando vengono premuti entrambi i pulsanti.

## Accompagnamento musicale

I comandi BASIC SOUND e PLAY permettono di migliorare la qualità dei vostri programmi accompagnandoli con un sottofondo musicale: nel Capitolo 10 esamineremo in dettaglio la capacità di produrre suoni del PC-IBM. C'è la possibilità di disporre un certo numero di note musicali in un "buffer di accompagnamento musicale": le note verranno poi suonate durante l'esecuzione del programma.
Per abilitare un programma BASIC a mantenere questo accompagnamento durante tutta l'esecuzione del programma potete utilizzare PLAY, controllato dalle istruzioni ON PLAY(*n*) GOSUB *linea*, PLAY ON, PLAY OFF e PLAY STOP.
L'argomento *n* è un intero che varia tra 1 e 32 ed indica il numero di note presenti nel buffer che creano l'evento di intercettamento: ogni volta che il numero di note diviene minore di *n*, viene intercettato l'evento PLAY e la routine relativa può inserire altre note nel buffer di accompagnamento musicale.

## Timer

Il TIMER permette di specificare un certo numero di secondi che devono trascorrere per chiamare la routine di intercettamento; l'istruzione ON TIMER(*n*) GOSUB *linea* ha come argomento il numero di secondi (da 1 a 86400=24 ore) che intercorrono tra due successivi eventi temporali. Le istruzioni TIMER ON, TIMER OFF e TIMER STOP permettono di abilitare e disabilitare l'evento quando necessario senza naturalmente interferire con l'inizializzazione di TIMER e TIME$.
Questo evento si rivela molto utile per i programmi che richiedono la misura di un intervallo di tempo: supponete infatti di avere un programma che proponga ad uno studente un test sulla storia; in 30 minuti lo studente deve rispondere al maggior numero possibile di domande. L'evento TIMER viene utilizzato per contare il tempo restante e visualizzarlo sullo schermo; allo scadere del tempo, il test finisce e viene valutato:

```
10 MIN=30:SEC=0
20 KEY OFF:CLS:LOCATE 25,1:PRINT "Minuti rimasti -";
22 TIMER ON
30 GOSUB 500
50 '    .
60 '    .    Parte del programma che visualizza il test
70 '    .
80 '    .
90 GOTO 30
```

```
500 COL=POS(0):ROW=CRSLIN ' Memorizza la posizione del
cursore
510 LOCATE 25,16
520 PRINT MIN; ":";SEC;' Visualizza il tempo rimasto
530 IF (MIN=0) AND (SEC=0) THEN RETURN 1000
540 ON TIMER (5) GOSUB 550
550 SEC=SEC-5: IF (SEC<0) THEN SEC=55:MIN=MIN-1
560 RETURN
570 '
580 '
999 ' Fine del test e visualizzazione del messaggio
1000 BEEP:CLS:PRINT"Il tempo disponibile per il test è
terminato"
```

**Interfaccia di comunicazione**

In un'interfaccia di comunicazione abilitata per l'intercettamento di eventi, si verifica un evento ogni volta che un carattere viene rilevato nel buffer di input dell'interfaccia: ciò significa che non appena un dispositivo esterno inizia ad inviare dati, si crea un evento. La routine di intercettamento deve perciò essere in grado di leggere il carattere per svuotare il buffer dell'interfaccia.

Le istruzioni di controllo degli eventi dovuti all'interfaccia sono simili a quelle già incontrate: COM(*n*) ON, con *n* uguale a 1 o a 2, serve per abilitare l'evento in una delle due interfacce; ON COM(*n*) GOSUB *linea* indica all'Advanced BASIC che cosa fare quando viene rilevato un evento; COM(*n*) OFF, come ON COM(*n*) GOSUB 0, disabilita l'interfaccia indicata; COM(*n*) STOP inibisce temporaneamente la capacità di rilevare eventi dell'interfaccia indicata. Se l'evento si verifica quando è in vigore l'istruzione COM(*n*) STOP, non viene riconosciuto subito, ma solo dopo che un'istruzione COM(*n*) ON sia stata eseguita.

Ecco un esempio di quanto detto finora:

```
      .
      .
      .
50 GOSUB 5000 ' Routine di inizializzazione dell'inte
rfaccia 1
60 COM(1) ON ' Abilita l'interfaccia 1
70 ON COM(1) GOSUB 2000 ' Se viene letto un carattere
questo viene elaborato
      .
      .
      .
300 ' Routine di inizializzazione dell'ora
```

```
310 COM(1) STOP 'Sospende l'intercettamento in modo c
he l'ora inserita sia accurata
320 CLS
330 LOCATE 10,20
340 INPUT "Inserire l'ora, HH:MM:SS:, in modo che qua
ndo premete ENTER sia esatta - " ; TEMP$
350 TIME$=TEMP$
360 FOR I = 0 TO 100
370 LOCATE 12,20 : PRINT "Sono le " TIME$;
    : NEXT I
380 CLS
390 COM(1) ON ' Riabilita l'interfaccia
400 RETURN
    .
    .
    .
2000 ' Routine per l'input da interfaccia
    .
    .
    .
2020 RETURN
    .
    .
    .
```

Osservate come l'intercettamento dei caratteri in input venga inibito alla linea 310, in modo che sia possibile definire l'ora con accuratezza; questa inibizione prevede che i dati vengano ricevuti abbastanza lentamente, o che il buffer sia sufficientemente grande, in modo da non perdere nessuno dei caratteri giunti durante l'operazione di inizializzazione dell'ora. Vedremo più in dettaglio nel Capitolo 11 come controllare l'intero processo di comunicazione.

## 6.5 Trattamento degli errori

Questo paragrafo è dedicato all'esame di alcuni tipi di errori propri del BASIC; vedremo anche come scrivere programmi in grado di trattare gli errori in modo da prevenire l'interruzione del programma.

### TIPI DI ERRORE

In BASIC sono definiti 78 errori che possono essere raggruppati nelle seguenti categorie: errori sintattici e matematici, errori su file in I/O, errori su periferiche in I/O ed errori di esecuzione.

Gli errori sintattici, del tipo SYNTAX ERROR (errore di sintassi) o DUPLICATE DEFINITION (doppia definizione), sono dovuti ad errori di stesura del programma; gli errori matematici, come DIVISION BY ZERO (divisione per zero), si verificano se il valore di una variabile o costante in un'espressione non è lecito per l'operazione in cui è coinvolto.
Gli errori di I/O su file, come BAD FILE NUMBER (numero del file non corretto), hanno relazione con il trattamento dei file da parte del BASIC, che esamineremo nel prossimo capitolo; gli errori di I/O con periferiche, come DISK FULL (il disco è pieno), riguardano i problemi incontrati dal PC nel colloquio con le periferiche.
Infine, gli errori di esecuzione, come CAN'T CONTINUE (non posso continuare), significano che il BASIC non può eseguire il programma così com'è scritto, o con i dati correnti.
Ogni errore BASIC è identificato da un numero, che potete trovare nell'Appendice C, insieme ad una breve spiegazione del tipo di errore.
Di solito, quando l'esecuzione di una linea provoca un errore, il BASIC visualizza un messaggio di spiegazione dell'errore e, a seconda dei casi, continua l'esecuzione del programma o ritorna al modo diretto.

## CONTROLLO DEGLI ERRORI

L'intercettamento di errori consente di far continuare un programma anche se si verifica un errore; anche se molto simile all'intercettamento di eventi, precedentemente trattato, l'intercettamento di errori ha su di questo una priorità: in altre parole, se si verifica un errore, l'intercettamento di eventi viene inibito fino al termine della routine di errore, mentre l'intercettamento degli errori non è mai inibito dalla routine di intercettamento di un evento.
Per abilitare questa procedura, si usa l'istruzione ON ERROR, con la seguente sintassi:

ON ERROR GOTO *linea*

Il parametro *linea* indica l'inizio della routine di intercettamento dell'errore. Osservate che non viene indicato il tipo di errore: tutti gli errori producono un salto alla stessa routine e diviene perciò compito della routine stessa identificare il tipo di errore rilevato.
Come per gli eventi, se *linea* è uguale a 0, l'intercettamento è disabilitato; la disabilitazione avviene, inoltre, durante l'esecuzione della routine. Nel caso in cui si verifichi un errore all'interno della routine di intercettamento, il BASIC visualizza il messaggio d'errore relativo e termina l'esecuzione del programma.
All'interno della routine potete usare la funzione ERR per determinare il

tipo di errore e la funzione ERL per determinare la linea che ha provocato l'errore; le due funzioni devono essere impiegate in questo modo:

*variabile*=ERR
*variabile*=ERL

ERR assume come valore il numero dell'errore ed ERL il numero della linea in cui l'errore si è verificato: potete utilizzare questi valori per scoprire esattamente quale sia l'errore e cosa fare per ripararvi; ad esempio:

```
      .
      .
      .
3000 IF ERR=27 THEN 3050
3010 IF ERR=24 THEN 3070
3020 IF ERL=(450 OR 600) THEN 3080
      .
      .
      .
```

L'istruzione RESUME conclude la routine di intercettamento; essa può essere utilizzata con le seguenti sintassi, a seconda delle necessità:

| | |
|---|---|
| RESUME [0] | L'esecuzione continua dall'istruzione che ha causato l'errore |
| RESUME NEXT | L'esecuzione continua dalla linea successiva a quella che ha causato l'errore |
| RESUME *linea* | L'esecuzione continua dalla linea indicata |

Ecco un programma che esemplifica il modo di trattare gli errori:

```
      .
      .
      .
3000 IF ERR=27 THEN 3050
3010 IF ERR=24 THEN 3070
3020 IF ERL=(450 OR 600) THEN 3080
3030 PRINT "Grave errore #" ERR " nella linea " ERL
3040 END
3050 LOCATE 25,1
3060 BEEP:PRINT "Accendete la stampante"
3070 RESUME
3080 PRINT "L'input non puo' essere piu' breve dell'e
sempio "
3090 RESUME 230
      .
      .
      .
```

Questa routine di intercettamento controlla due tipi d'errore, entrambi, in questo caso, connessi con il funzionamento della stampante. L'errore numero 27, OUT OF PAPER (mancanza di carta) può voler dire che la stampante è spenta o che manca la carta; l'errore numero 24, DEVICE TIMEOUT può significare in questo caso che la stampante non ha avuto tempo sufficiente per eseguire un cambio di pagina o un altro comando di controllo. La linea 3000 fa sì che venga visualizzato sullo schermo un messaggio esplicativo, mentre la linea 3010 fa in modo che il programma esegua un loop finché la stampante non è pronta: l'errore 27 viene in questo modo continuamente ripetuto, e perciò il messaggio rimane sullo schermo ed il segnale acustico risuona finché l'operatore non ha provveduto a correggerlo.
La linea 3020 controlla se l'errore è accaduto nelle linee 450 o 600: si presume cioè che la causa dell'errore possa essere dovuta all'input dell'operatore e lo si avvisa con un appropriato messaggio che viene visualizzato dalla linea 3080; l'esecuzione poi procede dalla linea 230, in cui viene ripetuto l'input.
Se l'errore non è contemplato dai test delle linee 3000, 3010 e 3020, la linea 3030 visualizza il numero dell'errore e la linea in cui si è verificato, in modo da dare all'operatore qualche idea della causa dell'interruzione.

## ERRORI DEFINITI DALL'UTENTE E SIMULATI

L'istruzione ERROR può essere usata sia per definire degli errori di propria scelta sia per simulare degli errori BASIC: può essere utile nel controllare un nuovo programma o nella revisione di uno vecchio. La sintassi dell'istruzione è:

ERROR *n*

Il parametro *n* è un'espressione numerica il cui valore è compreso nell'intervallo da 0 a 255. Quando viene eseguita, il BASIC simula l'errore indicato: in questo modo, se avete abilitato l'intercettamento degli errori, viene eseguita la routine di intercettamento; la funzione ERR usata all'interno della routine assumerà il valore *n*.
Se non è stato abilitato l'intercettamento, il BASIC visualizza il messaggio d'errore associato e di qui prosegue. Se non corrisponde ad uno degli errori definiti nel BASIC, viene visualizzato il messaggio UNPRINTABLE ERROR.

## 6.6 Ricerca degli errori nei programmi

Dopo che avete scritto un programma, solitamente lo mandate in esecuzione, ma spesso le cose non vanno come dovrebbero, qualsiasi sia la vostra precedente esperienza di programmazione. Vediamo perciò come compiere un *debugging* o caccia-all'errore di un semplice programma usando due strumenti software progettati per questo scopo: un tracciatore di programma e un controllore di variabili.

### I COMANDI TRON E TROFF

I comandi TRON (*trace on*) e TROFF (*trace off*) vengono usati per riprodurre il flusso del programma istante per istante mentre viene eseguito. Quando il programma è pronto per essere controllato, potete farne una copia con la stampante in modo da seguire, una linea dopo l'altra, il cammino del programma come viene visualizzato; quindi, in modo diretto, inviate il comando:

```
TRON
```

e durante l'esecuzione verrà visualizzato il numero di ogni linea eseguita. Se lo schermo è pieno di numeri e voi volete sospendere temporaneamente l'esecuzione del programma, premete i tasti CTRL e NUM LOCK; quando sarete di nuovo pronti, un qualsiasi tasto farà riprendere l'esecuzione ed i numeri continueranno ad apparire.
Se il vostro programma deve anche visualizzare dei dati sullo schermo, l'output sarà inframezzato dai numeri di linea: nel caso in cui il programma sposti il cursore esplicitamente per mezzo dell'istruzione LOCATE, o contenga delle istruzioni CLS, potreste perdere parte dei numeri di linea visualizzati.
Durante il tracciamento di un programma potete tornare al modo diretto premendo CTRL BREAK (a meno che il programma non sia già terminato da solo) ed inviando il comando:

```
TROFF
```

### CONTROLLO DEI VALORI DELLE VARIABILI

L'istruzione PRINT può esservi d'aiuto nel tenere sotto controllo i valori delle variabili in punti critici del programma. Se un loop non funziona correttamente, potete inserire un'istruzione PRINT che visualizzi il valo-

re dell'indice, per controllare a che punto si verifica l'errore; se interrompete (BREAK) un programma, potete ancora usare PRINT in modo diretto per esaminare il valore corrente delle variabili del programma.

## COME ESEGUIRE IL DEBUGGING DI UN PROGRAMMA

Supponete di avere in memoria questo programma:

```
5 Z=0
10 FOR T=0 TO 30 STEP 2
20    S=T MOD 2
30    IF S THEN Z=Z+1
40 NEXT T
50 IF Z=15 THEN BEEP
60 END
```

Lo scopo di questo programma è di generare un beep: durante l'esecuzione, però, non accade nulla. Per vedere cosa accade, inviamo il comando TRON e rilanciamo il programma. Il risultato è il seguente:

```
Ok
TRON
Ok
RUN
 [5][10][20][30][40][20][30][40][20][30]
[40][20][30][40][20][30][40][20][30][40]
[20][30][40][20][30][40][20][30][40][20]
[30][40][20][30][40][20][30][40][20][30]
[40][20][30][40][20][30][40][20][30][40]
[50][60]
Ok
```

Ognuno dei numeri tra parentesi quadre è il numero della linea che viene eseguita. Tutto quello che potete dire a questo punto, però, è che il programma esegue un loop tra le linee 20 e 40, come del resto vi aspettavate. Il passo successivo è quello di vedere cosa accade alla variabile Z. Listate il programma ed inserite l'istruzione PRINT in questo modo:

```
Ok
LIST
5 Z=0
10 FOR T=0 TO 30 STEP 2
20    S=T MOD 2
30    IF S THEN Z=Z+1
40 NEXT T
```

```
50 IF Z=15 THEN BEEP
60 END
Ok
35 PRINT "Z =" Z
```

Eseguite ora la nuova versione:

```
Ok
RUN
[5][10][20][30][35]Z = 0
[40][20][30][35]Z = 0
[40][20][30][35]Z = 0
[40][20][30][35]Z = 0
[40][20][30][35]Z = 0
[40][20][30][35]Z = 0
[40][20][30][35]Z = 0
[40][20][30][35]Z = 0
[40][20][30][35]Z = 0
[40][20][30][35]Z = 0
[40][20][30][35]Z = 0
[40][20][30][35]Z = 0
[40][20][30][35]Z = 0
[40][20][30][35]Z = 0
[40][20][30][35]Z = 0
[40][20][30][35]Z = 0
[40][50][60]
Ok
```

Come potete vedere, il valore di Z non è mai incrementato; guardando il programma più attentamente vi accorgete che, perché Z sia incrementato, S dev'essere uguale a 1, ed S è posto uguale a 1 solo quando T è dispari; ma T viene incrementato di 2 ad ogni passo del loop e, poiché parte da 0, sarà sempre pari. È necessario quindi cambiare l'intervallo di variabilità di T come segue:

```
10 FOR T=1 TO 30 STEP 2
```

Il programma è eseguito di nuovo e si ottiene:

```
Ok
RUN
[5][10][20][30][35]Z = 1
[40][20][30][35]Z = 2
[40][20][30][35]Z = 3
[40][20][30][35]Z = 4
[40][20][30][35]Z = 5
[40][20][30][35]Z = 6
```

```
[40][20][30][35]Z  =  7
[40][20][30][35]Z  =  8
[40][20][30][35]Z  =  9
[40][20][30][35]Z  =  10
[40][20][30][35]Z  =  11
[40][20][30][35]Z  =  12
[40][20][30][35]Z  =  13
[40][20][30][35]Z  =  14
[40][20][30][35]Z  =  15
[40][50][60]
Ok           <-------------------------- Il PC suona
```

Come mostrato dall'istruzione PRINT, il valore di Z viene ora incrementato, ed il PC emette il suono richiesto.

Capitolo

# I file BASIC

# 7

In questo capitolo esamineremo il modo di creare raccolte di dati, immagazzinarle su floppy disk o altri dispositivi esterni e infine rileggere e caricare i dati memorizzati in precedenza. Queste raccolte di dati vengono dette *file BASIC*: costituiscono parte integrante dei programmi che utilizzano grandi quantità di dati o che devono comunicare con dispositivi esterni.

## 7.1 I file

Un file è perciò un insieme di dati associato con un particolare dispositivo fisico (un drive, per esempio); viene utilizzato per immagazzinare programmi su disco e più tardi ricaricarli in memoria.
Oltre che programmi, i file possono contenere dati a cui i programmi devono avere accesso. Un programma di inventario potrebbe creare una grande tabella o un elenco contenente tutte le merci disponibili in magazzino; la lista verrebbe memorizzata in un file su disco per poter essere riletta o aggiornata successivamente.
I file possono contenere o programmi o dati, mai entrambi: esaminiamo ora in dettaglio questi tipi di file.

### FILE DI PROGRAMMA

I file di programma usualmente contengono una rappresentazione ASCII o binaria dei programmi scritti per il PC; questi file, creati dal BASIC, o

altro linguaggio ad alto livello, oppure scritti in Assembler, costituiscono ciò che in realtà viene utilizzato dal PC mentre esegue il programma.
Se battete, infatti, RUN "CAMPIONI.BAS" in risposta al prompt del BASIC, avvisate il BASIC di caricare il file "CAMPIONI.BAS" dal drive di default; il comando RUN, inoltre, presume che il file "CAMPIONI.BAS" sia un file di programma e perciò cerca nel file i numeri di linea seguiti da istruzioni eseguibili.

## FILE DI DATI

I file di dati, invece, contengono informazioni create da un programma o destinate ad essere elaborate dallo stesso. Un programma word processor elabora un file di dati riempito con un testo: l'operatore può visualizzare la versione più recente del testo, modificarla a suo piacimento e, se crede, memorizzarla come un altro file separato.
Questo tipo di file è molto utile quando il programma richiede o produce una grande quantità di dati: di solito, infatti, i file usati come input o output di un programma sono file di dati e non di programmi.

## DISPOSITIVI PER I FILE

Diversi sono i dispositivi che possono inviare file come input per un programma o riceverli come output, o che compiono entrambe le operazioni. Dovete perciò specificare il dispositivo che vi interessa quando nominate un file; il BASIC è in grado di riconoscere numerosi nomi di dispositivi come parte di una specificazione di file valida. (Vedi il paragrafo 4.8 per maggiori informazioni sulle specificazioni). I dispositivi che supportano file BASIC sono elencati nella Tabella 7.1.

**Tabella 7.1** Dispositivi per file BASIC

| Dispositivo | Nome | Input | Output |
|---|---|---|---|
| Drive A: | A: | X | X |
| Drive B:, C:, D: | B:, C:, D: | X | X |
| Tastiera | KYBD: | X | |
| Schermo | SCRN: | | X |
| Stampante #1 | LPT1: | | X |
| Stampante #2 | LPT2: | | X |
| Stampante #3 | LPT3: | | X |
| Interfaccia per comunicazioni asincrone #1 | COM1: | X | X |
| Interfaccia per comunicazioni asincrone #2 | COM2: | X | X |

Un dischetto può contenere diversi file: ogni file è identificato con il nome, che dev'essere suo proprio (ad esempio B: TESTPROG.BAS), e può essere usato sia come sorgente che come destinazione dei dati; se caricate un programma da un file su disco, usate il file come sorgente, mentre se salvate un programma su disco usate il file come destinazione.
Alcuni dispositivi, però, possono essere associati con un unico file e perciò con un unico nome di file; inoltre, alcuni dispositivi possono essere usati esclusivamente per file in input o in output. Lo schermo può essere visto dal BASIC come un file di nome SCRN:; questo file funziona solo da destinazione per i dati, cioè come file di output. L'esempio contrapposto è la tastiera che, usata come file KYBD:, agisce unicamente come file di input: non potete inviare alcuna informazione alla tastiera.
Altri dispositivi possono essere usati come file BASIC senza che ne sia specificato il nome: non dovete infatti indicare esplicitamente la tastiera o lo schermo come file di input e di output quando usate istruzioni come INPUT o PRINT; potete trovare molti esempi di uso di input ed output strutturato come file senza per questo che il dispositivo sia specificato esplicitamente.

## FILE SU DISCO

I file su disco possono essere suddivisi in tre categorie fondamentali: file *sequenziali*, file *ad accesso diretto* e file *pseudo-sequenziali*. La principale differenza tra queste categorie sta nel modo in cui i dati vengono memorizzati e riletti.
I dati in un file sequenziale sono sempre letti uno dopo l'altro; non potete saltare avanti e indietro in un file sequenziale, né in input né in output. In un file ad accesso diretto, invece, le informazioni sono raggiungibili in qualsiasi ordine: potete portarvi alla locazione desiderata all'interno del file sia in scrittura (output) sia in lettura (input). Ai file pseudo-sequenziali si può accedere come ai file sequenziali anche se sono memorizzati come i file ad accesso diretto: questo ha alcuni vantaggi ed alcuni svantaggi, come vedremo più avanti.
Il principale vantaggio dei file sequenziali è che sono facili da usare: sono poche le operazioni preliminari da compiere per predisporre un file sequenziale ed il processo di reperimento dei dati è semplice. I file ad accesso diretto, d'altra parte, occupano meno spazio sul disco e sono più versatili, in quanto potete raggiungere direttamente un punto qualsiasi del file.
I file pseudo-sequenziali costituiscono un compromesso tra i due tipi precedenti: memorizzano i dati in un formato "compresso", come i file ad accesso diretto, ma richiedono più operazioni preliminari rispetto ai file sequenziali. Accedere ai dati di un file pseudo-sequenziale è molto simile

ad accedere ai file sequenziali: non potete raggiungere una posizione arbitraria nel file, ma dovete leggere un termine dopo l'altro, in sequenza.

## 7.2 File sequenziali

Un file sequenziale altro non è che una serie di caratteri o di valori senza alcun formato intrinseco.
Quando leggete un file sequenziale dovete sempre partire dall'inizio: non potete iniziare da un punto arbitrario. Quando scrivete su di un file sequenziale avete due possibilità: potete cominciare dall'inizio del file (perdendo ciò che eventualmente vi fosse già scritto) o aggiungere i vostri dati alla fine del file già scritto; non potete però inserire dati a partire da un punto arbitrario del file.
Man mano che i dati vengono aggiunti ad un file sequenziale, vengono immagazzinati nello stesso ordine in cui sono inviati. Supponete di aver un file che contiene tre dati:

```
DATO 1 ←————————— Inizio del file sequenziale
DATO 2
DATO 3 ←————————— Fine del file sequenziale
```

Al crescere del file, un puntatore segnala la successiva locazione disponibile: se aggiungete altri dati, questi andranno a finire automaticamente nella locazione indicata dal puntatore, cioè alla fine del file. Aggiungendo un quarto termine al file precedente si ottiene:

```
DATO 1
DATO 2
DATO 3
DATO 4 ←————————— Nuova fine del file sequenziale
```

Quando iniziate a leggere un file sequenziale il primo dato è quello che proviene dall'inizio del file e i successivi vengono via via recuperati nello stesso ordine in cui erano stati scritti all'origine:

```
DATO 1 ←————————— Inizio del file sequenziale;
                  questo è il primo dato letto
DATO 2
DATO 3
DATO 4 ←————————— Fine del file sequenziale
```

Tutti i dispositivi per file BASIC sono in grado di funzionare da supporto per la struttura sequenziale dei file: la maggior parte di essi, infatti, è intrinsecamente sequenziale.
Vediamo ora come creare un file sequenziale su disco in BASIC; la trattazione è valida per tutti i dispositivi per file, tranne l'interfaccia per comunicazioni asincrone (ACA): questa ed i file sequenziali di comunicazione verranno discussi nel Capitolo 11.

## ACCESSO AI FILE SEQUENZIALI

La procedura per accedere ai file sequenziali si svolge in passi successivi: innanzitutto il file dev'essere *aperto*.
Aprendo un file indicate al BASIC di riservare una parte di memoria sia per trasferire dati da e al dispositivo sul quale il file risiede, sia per seguire la traccia del processo.
Dopo che è stato aperto, il file è disponibile per l'output (scrivere dati sul file) o per l'input (leggere dati dal file): non potete usare un file sequenziale contemporaneamente per l'output e l'input. Quando indicate che i dati sono destinati al file dovete anche specificare se essi vanno aggiunti alla fine o all'inizio del file stesso: ricordate che, nel secondo caso, tutti i dati contenuti precedentemente nel file andranno irrimediabilmente persi.
Una volta che avete terminato le operazioni con il file dovete ricordarvi di chiuderlo. Chiudendo il file, istruite il BASIC a scrivere nel file tutti i dati che ancora restano in memoria e a rilasciare lo spazio di memoria riservato per controllare l'accesso al file.

## COME APRIRE UN FILE SEQUENZIALE

L'istruzione OPEN crea un canale di comunicazione con un file residente nel dispositivo indicato. Sono due le sintassi possibili dell'istruzione:

OPEN "*specfile*" [FOR *modo*] AS [#] *numfile* [LEN=*lungrec*]

oppure

OPEN *modo2*, [#] *numfile*, "*specfile*"

*specfile* è l'indicazione del file (*d:*[*cammino*] *nomefile.est*) come presentata nel Capitolo 4. Se il file sequenziale si trova su un dispositivo diverso da un disco o da una cassetta, la specificazione consiste nel solo nome del dispositivo, che trovate nella Tabella 7.1.
I parametri *modo* e *modo2* sono espressioni di tipo stringa che indicano

in che modo il programma deve avere accesso al file sequenziale. I parametri sono definiti come segue:

| Modo | Modo2 | Tipo di accesso |
|---|---|---|
| OUTPUT | O | Output sequenziale |
| INPUT | I | Input sequenziale |
| APPEND | non esiste | Aggiunta sequenziale ad un file già esistente |

I dati originali vengono distrutti se un file già esistente viene aperto con l'opzione OUTPUT: le successive istruzioni di output scriveranno i dati a partire dall'inizio del file cancellato; il modo APPEND viene usato per aggiungere dati ad un file già esistente, in modo che i nuovi dati vengano scritti a partire dalla fine di quelli precedenti.
Il parametro *numfile* è un'espressione intera utilizzata dalle successive istruzioni di accesso al file per identificarlo: il valore di *numfile* può variare da 1 a 4 nel Cassette BASIC e da 1 a 3 (valore di default) nel Disk e nell'Advanced BASIC. L'opzione /F: del comando BASIC può essere utilizzata per cambiare il massimo numero di file che possono essere aperti (al massimo 15) nel Disk o nell'Advanced BASIC. La lunghezza del record, definita dal parametro opzionale LEN=*lungrec*, non viene di solito usata per i file sequenziali.
Ecco un esempio di istruzione OPEN per un file sequenziale:

```
300 OPEN "A:INVENT.DAT" FOR APPEND AS #6
```

L'istruzione apre il file INVENT.DAT come file sequenziale sul disco inserito nel drive A:. Si può solo scrivere nel file ed ogni nuovo dato viene aggiunto al termine di quelli già presenti; ogni istruzione di output si riferirà a quel file con il numero 6.
Un altro esempio di istruzione OPEN:

```
20 OPEN "I", 2, "A:CONTGEN.FAT"
```

che apre il file CONTGEN.FAT come file sequenziale del disco inserito nel drive A:. Questo file può solo essere letto; le istruzioni di input si riferiscono al file con il numero 2.

## COME SCRIVERE SU UN FILE SEQUENZIALE

Dopo avere aperto un canale per l'output a un file sequenziale, si possono usare le seguenti istruzioni per inviare informazioni al file:

PRINT#
PRINT# USING
WRITE#

Queste istruzioni funzionano quasi esattamente come le loro corrispondenti per l'output su schermo: la principale differenza sta nel fatto che queste ultime contengono anche il numero di un file e scrivono dati su questo, invece che sullo schermo. Per scrivere dati nel file numero 2, perciò, usiamo istruzioni come:

```
200 PRINT#2, 43.33;VALORE ' VALORE = 4
210 PRINT#2,USING "_@##.#^^^^";923800,6.555E-5
```

Dopo l'esecuzione, nel file 2 si troverà:

```
43.33 4 @92.4E4 @65.6E-4
```

Spazio   A capo e ritorno carrello

Anche l'istruzione WRITE# si comporta come l'equivalente per lo schermo: quando è utilizzata per l'output ad un file sequenziale, i dati numerici devono essere separati da virgole e le stringhe da virgolette.
Spazi, virgole, virgolette, ritorni carrello e a capo sono segni delimitatori che separano un termine dall'altro in modo da stabilire con esattezza come il dato dev'essere letto. Per comprendere l'effetto di questi caratteri, immaginate questa situazione: i dati vengono scritti in un file sequenziale per mezzo delle istruzioni:

```
50 POSIZIONE$="Presidente"
60 COGNOME$="Agnelli"
70 PRINT #3, POSIZIONE$;COGNOME$
```

Nel file numero 3 saranno scritti perciò:

```
PresidenteAgnelli
```

Osservate che questo dato appare come un'unica stringa: se volessimo usare questo file come input, non riusciremmo facilmente a separare i due termini.
Per memorizzare questi due dati in modo che sia semplice distinguerli l'uno dall'altro, nell'istruzione PRINT dobbiamo inserire un delimitatore, come una virgola che separi le due stringhe nell'istruzione PRINT#:

```
70 PRINT #3,POSIZIONE$;",";COGNOME$
```

Dopo l'esecuzione di quest'ultima istruzione, il file apparirà così:

```
Presidente,Agnelli
```

Le stringhe ora sono facilmente distinguibili, data la presenza della virgola tra i due dati.

## COME LEGGERE DA UN FILE SEQUENZIALE

Un file sequenziale che sia stato aperto per l'input può essere letto con una delle seguenti istruzioni:

INPUT#
LINE INPUT#
INPUT$

Le due istruzioni INPUT# e LINE INPUT# funzionano con i file sequenziali proprio come le istruzioni INPUT e LINE INPUT funzionano con la tastiera: se infatti utilizzate la tastiera come file sequenziale, potete usare questa versione dell'istruzione INPUT:

```
40 OPEN "I", #1, "KYBD:"
50 LINE INPUT #1, KEYENTRY$
```

L'istruzione INPUT#, con la sintassi

INPUT #*numfile*, *variabile* [, *variabile*]...

legge i dati dalla posizione corrente nel file sequenziale e li assegna alle corrispondenti variabili che appaiono nell'istruzione; il tipo di dati da leggere deve concordare con il tipo delle variabili a cui vanno assegnati i dati, altrimenti si incorre in un errore del tipo TYPE MISMATCH.
Quando l'istruzione INPUT# deve leggere dei dati numerici, tutti gli spazi iniziali, i ritorni carrello e a capo vengono trascurati, in quanto questi caratteri funzionano solo da delimitatori con i dati di tipo numerico: INPUT# inizia a leggere il dato a partire dal primo carattere che sia riconosciuto non essere un delimitatore.
L'istruzione INPUT# può anche servire per leggere dati di tipo stringa: di nuovo, l'inizio della stringa è considerato il primo carattere che non sia uno spazio, un ritorno carrello o un a capo; se le virgolette sono il primo carattere, tutti i caratteri seguenti fino alle successive virgolette vengono considerati parte di un'unica stringa, altrimenti la stringa cor-

rente termina con la prima virgola, o il primo carattere di ritorno carrello o di a capo, oppure al 255-esimo carattere letto.
Supponiamo che i seguenti termini facciano parte di un file sequenziale:

```
Mario,Rosso Maria,Rossi
```

La seguente istruzione interpreterà i caratteri come costituenti tre stringhe:

```
90 INPUT #8, NOME1$, NOME2$, NOME3$
```

Per mostrare l'uso delle virgolette come delimitatori, consideriamo il programma:

```
150 POSIZIONE$="Lucio"
160 NOME$="Dalla,Sandro,Pertini"
170 PRINT#3,POSIZIONE$;NOME$
```

A questo punto l'istruzione:

```
INPUT#3,POSIZIONE$,NOME$
```

assegnerà alla variabile POSIZIONE$ la stringa "LucioDalla" ed alla variabile NOME$ la stringa "Sandro". Per assicurare la correttezza della scrittura, l'istruzione PRINT# dovrebbe delimitare le stringhe in questo modo (osservate che le virgolette devono essere inserite usando la funzione CHR$(34)):

```
170 PRINT#3,CHR$(34);POSIZIONE$;CHR$(34);
            CHR$(34);NOME$;CHR$(34)
```

così da scrivere nel file numero 3:

```
"Lucio""Dalla,Sandro,Pertini"
```

Una successiva istruzione INPUT# interpreterà le virgolette come delimitatori e questo permetterà di leggere correttamente le due stringhe.
L'istruzione LINE INPUT# interpreta i dati provenienti da un file sequenziale in modo leggermente differente. La sintassi dell'istruzione è:

LINE INPUT #*numfile, varstr*

L'unico delimitatore che LINE INPUT# è in grado di riconoscere è la sequenza ritorno carrello/a capo; viene perciò utilizzata questa istruzione

quando è importante conservare esattamente spazi e virgole all'interno della stringa.
L'istruzione INPUT$, al contrario della precedente, non riconosce alcun delimitatore all'interno di un file sequenziale; la sua sintassi è la seguente:

X$=INPUT$(*n*[,[#]*numfile*])

dove *n* indica il numero di caratteri che verranno letti dal file numero *numfile* ed assegnati alla stringa X$. Se *numfile* non viene specificato, i caratteri vengono presi dalla tastiera e in questo caso INPUT$ agisce in modo molto simile all'istruzione INKEY$.
Concludendo, INPUT$ è usata per leggere un determinato numero di caratteri dal file indicato, esattamente come essi appaiono nel file stesso. Come vedremo nel Capitolo 11, INPUT$ viene spesso usata insieme con un file di comunicazione; quando usate l'istruzione INPUT$ con un file su disco, dovete conoscere in precedenza la lunghezza esatta di ogni termine.

## COME CONOSCERE LO STATO DI UN FILE SEQUENZIALE

Per mezzo di tre funzioni del BASIC potete conoscere alcune informazioni che riguardano un dato file sequenziale. Le funzioni e le loro sintassi, sono le seguenti:

X=LOF(*numfile*)
X=LOC(*numfile*)
X=EOF(*numfile*)

La funzione LOF restituisce la lunghezza, in byte, del file corrispondente al numero *numfile*. Il seguente esempio mostra uno degli impieghi della funzione LOF: qui si chiede la lunghezza di un file che deve essere elaborato ed il programma esegue un calcolo, in base a questa informazione, per informare l'operatore del tempo necessario a completare il lavoro.

```
        .
        .
        .
1010 INPUT "Nome del file da elaborare (drive A:) ";N$
1020 M$="A:"+N$ ' Aggiunge l'identificatore del drive
A: al nome del file inserito dall'utente
1030 OPEN "I",#5,M$ ' Apre in input, attraverso #5, il
il file contenuto in M$
```

```
1040 LUNGHFILE=LOF(5)/9  'Ogni termine del file e'
un numero in doppia precisione,  cioe' di 8 byte
per numero, piu' un byte per un delimitatore (spaz
io) aggiunto quando il file e' stato creato
1050 TEMPO=LUNGHFILE*1.355  'In media servono 1.35
5 secondi per elaborare un singolo termine
1060 PRINT "Il file, " N$ " sara' elaborato in cir
ca "TEMPO" minuti. Prenditi una sedia e riposati."
     .
     .
     .
```

La funzione LOC restituisce la locazione corrente all'interno del file specificato, riferita all'ultima volta in cui è stato aperto, in termini di blocchi di dati di 128 byte; in pratica, cioè, LOC vi dice quanti byte, approssimati per eccesso a un multiplo di 128, sono stati letti o scritti da un file sequenziale dal momento in cui è stato aperto per l'ultima volta. Se il file è stato aperto per l'output, LOC restituisce il numero totale di blocchi di 128 byte contenuti nel file; se il file è stato aperto per l'aggiunta, LOC restituisce il numero di blocchi di 128 byte aggiunti.
Ecco ora un esempio dell'uso della funzione LOC: il tasto funzione F7 permette all'operatore di richiedere lo stato corrente di un file; quando viene premuto F7, viene visualizzata la quantità di elementi del file indicato dalla variabile FILECORRENTE ancora da elaborare; segue un loop di ritardo in modo che l'operatore possa leggere l'informazione e infine la routine riprende da dove era stata interrotta. Questa routine presuppone che il file sia riempito con elementi lunghi 16 byte l'uno.

```
          .
          .
          .
2000 ON KEY(79 GOSUB 7000
2010 KEY 7, "STATO"
2020 KEY (7) ON
          .
          .
7000 PRINT "RESTANO ";LOC(FILECORRENTE)*(128/16);" ELE
MENTI NEL FILE"
7010 FOR X=0 TO 500 :  NEXT : RETURN
          .
          .
          .
```

La funzione EOF indica se è stata raggiunta la fine di un file: se l'ultimo termine acquisito è l'ultimo termine del file, il valore della variabile è vero (EOF= −1), altrimenti è falso (EOF=0). Nell'esempio seguente viene

letto un carattere per volta finché EOF(1) non indica che è stata raggiunta la fine del file:

```
      .
      .
      .
370 X$=INPUT$(1,#1)          'Prende un carattere dal
file
380 GOSUB 1000               'Elabora il carattere
390 IF EOF(1) THEN 410
400 GOTO 370                 'Prende il carattere suc
cessivo
410 ...
      .
      .
      .
```

## COME CHIUDERE UN FILE SEQUENZIALE

L'ultimo passo di tutta la procedura consiste nel chiudere il file: dovete sempre chiudere un file sequenziale dopo averlo aperto, sia che abbiate terminato di usarlo (ad esempio, alla fine di un programma) sia che intendiate usarlo in un altro modo (lettura, scrittura o aggiunta). Il file non correttamente chiuso, può perdere alcuni dati.
Per questa operazione utilizzate l'istruzione CLOSE, con la sintassi:

CLOSE [[#] *numfile* [,[#] *numfile*]...]

Se non vengono indicati i parametri *numfile*, tutti i file aperti vengono chiusi; altrimenti, l'istruzione CLOSE chiude solo i file il cui numero è specificato.
Chiudendo il file si provoca la scrittura del contenuto di un buffer che il BASIC riserva per quel file nel dispositivo in cui il file è stato aperto.
Molte altre istruzioni provocano la chiusura di tutti i file aperti; queste sono:

END
NEW
RESET
SYSTEM

## UN ESEMPIO DI ACCESSO AD UN FILE SEQUENZIALE

Ora sfruttiamo alcuni dei concetti descritti per mostrare come creare e leggere file sequenziali.

Nella Figura 7.1 è riportato un programma che crea un file sequenziale di stringhe sul disco inserito nel drive A:. Il file viene dapprima aperto per la lettura ed in seguito riempito con la successione di stringhe che vengono battute sulla tastiera; viene chiuso quando si batte la sola lettera Q ed infine riaperto per la scrittura. Lo schermo viene utilizzato come file sequenziale per la visualizzazione del programma: questo infatti legge i dati dal file su disco e li trasferisce sullo schermo.

```
10  '***ESEMPIO DI FILE SEQUENZIALE***
20  '
30  '   Questo programma crea e poi riempie un file se
quenziale
40  ' sul disco inserito nel drive A. I dati nel file
sono stringhe
50  ' inserite da tastiera. Dopo aver completato il fi
le,
60  '  il programma consente di esaminare le stringhe
una per
70  ' volta. Osservate come lo schermo sia utilizzato
come un
80  ' file sequenziale per output,proprio come se foss
e stata
90  ' usata una normale istruzione PRINT .
100 ' Questo programma richiede o il Disk BASIC o l'Ad
vanced
110 ' BASIC ed un disco gia' formattato nel drive A.
120 CLS:KEY OFF
130 PRINT "INSERIRE IL DISCO PER IL FILE NEL DRIVE A"
140 PRINT "PREMERE UN TASTO QUALSIASI PER CONTINUARE"
150 IF INKEY$="" THEN 150
160 CLS
170 OPEN "A:TESTSEQ.DAT" FOR OUTPUT AS 1 'Apre un file
 sequenziale nel drive A
180 PRINT "INSERIRE LA STRINGA DESTINATA AL FILE."
190 INPUT "SE VOLETE TERMINARE IL FILE, PREMETE 'Q'.",
A$    '   Stringa di input da inserire nel file
200 IF A$="Q" THEN 260
210 PRINT #1, A$ 'Output sul file #1: "A:TESTSEQ.DAT"
220 GOTO 180   ' Richiede un nuovo inserimento
230 '
240 ' Ora trasferisce i dati dal file sullo schermo
250 '
260 CLOSE ' Deve chiudere il file sequenziale prima di
 poterlo usare come input
270 CLS
280 PRINT "ORA POTETE VEDERE TUTTI GLI ELEMENTI DEL FI
LE UNO PER VOLTA."
290 PRINT
300 PRINT "PER VEDERE L'ELEMENTO SUCCESSIVO, PREMETE E
NTER."
```

(continua)

```
310 PRINT
320 IF INKEY$="" THEN 320
330 OPEN "A:TESTSEQ.DAT" FOR INPUT AS 1 ' Riapre il fi
le come input per il programma
340 OPEN "scrn:" FOR OUTPUT AS 2  'Adesso usa lo scher
mo come se fosse un file
350 INPUT #1, R$ 'Prende il successivo record dal file
 "A:TESTSEQ.DAT"
360 PRINT #2, R$  'Output sul file-schermo
370 IF EOF(1) THEN 400  ' Controlla la presenza di un
segnale di end-of-file nel file di input
380 IF INKEY$="" THEN 380
390 GOTO 350  'Prende il successivo record
400 CLOSE ' E' finito; chiude i file
410 PRINT
420 PRINT "Abbiamo finito!!"
430 END
```

**Figura 7.1** Esempio di accesso ad un file sequenziale

## 7.3 File ad accesso diretto

La caratteristica distintiva dei file ad accesso diretto o casuale è quella di possedere una struttura intrinseca prestabilita: i dati che vengono scritti in uno di questi file vengono organizzati in unità dette *record*; ogni record di un file ad accesso diretto contiene uno o più dati separati posti nel record con un ordine prestabilito.
I file ad accesso diretto vengono letti o scritti un record per volta: questi record sono numerati, in modo che ci si possa riferire ad uno di essi per mezzo del suo numero e raggiungerlo così immediatamente. Se volete leggere un determinato termine, potete raggiungere il record che lo contiene senza dover prima leggere tutti quelli che lo precedono: questa è la principale differenza tra questa struttura e quella dei file sequenziali.

### COME ACCEDERE AD UN FILE AD ACCESSO DIRETTO

Prima di poter accedere ad un file dovete aprire un canale che vi metta in comunicazione con il file ad accesso diretto, assegnando un numero al nome del file; l'apertura del file predispone anche in memoria un buffer per file ad accesso diretto, che serve da collegamento tra il programma ed il file.
Prima di utilizzare questo buffer dovete specificare il formato dei record che intendete usare: in questo caso, infatti, sono ben definiti lunghezza e ordine dei termini che compongono ogni singolo record.

I dati destinati al file vengono posti nel buffer fino a formare un record: quando tutti i termini del record sono nel buffer, il record intero viene trasferito al disco in cui risiede il file.
Analogamente, per recuperare i dati da un file, viene caricato un intero record dal file nel buffer da cui il programma legge i termini richiesti. Osservate che, una volta aperto, un file ad accesso diretto può essere usato sia in lettura che in scrittura, contrariamente a quanto accade per i file sequenziali.
Il processo ora spiegato è schematizzato nella Figura 7.2.

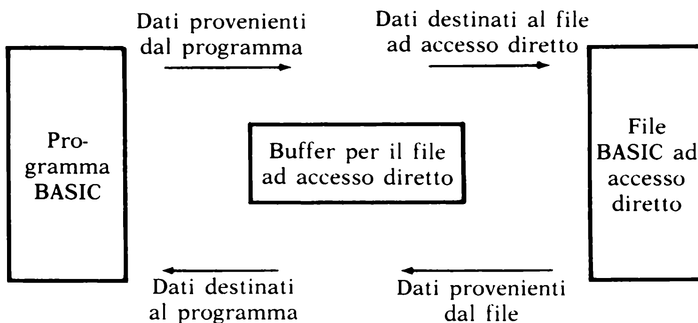



**Figura 7.2** File ad accesso diretto

Dopo aver terminato il suo compito, il file ad accesso diretto deve essere chiuso: ogni dato presente nel buffer viene in questo modo scritto nel file e viene liberata quella parte di memoria riservata per il controllo del processo.

## COME APRIRE UN FILE AD ACCESSO DIRETTO

Se volete abilitare all'uso un file ad accesso diretto, dovete innanzitutto aprirlo, come abbiamo appena visto: la sintassi dell'istruzione OPEN è la seguente

OPEN "*specfile*" [FOR *modo*] AS [#] *numfile* [LEN=*lungrec*]

oppure

OPEN "R",[#] *numfile*, "*specfile*" [,*lungrec*]

Non dovete specificare se il file vi serve per la scrittura o la lettura, poiché questi file possono essere usati contemporaneamente per entrambe.

Il parametro *specfile* dev'essere un'espressione di tipo stringa contenente un nome valido di file; osservate, però, che gli unici dispositivi che consentono l'uso di file ad accesso diretto sono i disk drive (A:, B:, C: o D:); *numfile* è un'espressione intera come quella definita per i file sequenziali. Il valore di *lungrec* non può superare quello definito dall'opzione /S:*b* al caricamento del BASIC, dal momento che /S:*b* stabilisce la massima lunghezza raggiungibile da un buffer per file ad accesso diretto. Il valore di default per *b* è 128 byte, mentre il valore massimo è 32767.

## COME DEFINIRE IL FORMATO DI UN RECORD

Ogni record di un file ad accesso diretto ha un formato fisso che deve essere stabilito a priori, per mezzo del parametro *lungrec* dell'istruzione OPEN; *lungrec* è un'espressione a valori interi, che vanno da 1 a 32767 (32K), ed indica la lunghezza del record in byte (un byte per ogni carattere); se il parametro è omesso, per default la lunghezza del record è di 128 byte.
Vi mostriamo ora due modi equivalenti per aprire un file ad accesso diretto di nome LIBRI.NUM sul disco inserito nel drive A:, per assegnare al file il numero 7 e ad ogni record la lunghezza di 40 caratteri:

```
300 OPEN "A:LIBRI.NUM" AS #7 LEN=40

300 OPEN "R", #7, "A:LIBRI.NUM", 40
```

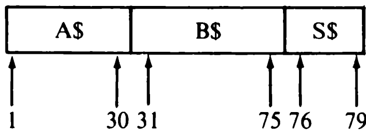
Per definire il formato del record, poi, potete usare l'istruzione FIELD, con la sintassi:

FIELD [#] *numfile*, *dim* AS *varstr* [,*dim* AS *varstr*]...

Il parametro *numfile* indica il file a cui si riferisce l'istruzione FIELD: questo numero deve corrispondere a quello di una precedente istruzione OPEN.
La parte rimanente dell'istruzione contiene una serie di variabili di tipo stringa, che definiscono in che modo i termini devono essere posti nel record: ogni stringa corrisponde ad un diverso termine. La lunghezza totale delle stringhe, cioè la somma di tutti i parametri *dim*, non deve superare la lunghezza del record precedentemente definita, altrimenti si verificherà un errore FIELD OVERFLOW.
Esaminate il seguente esempio di istruzione FIELD:

```
20 FIELD #4, 30 AS A$, 45 AS B$, 4 AS S$
```

I primi 30 caratteri di ogni record del file numero 4 vengono riservati per A$, i successivi 45 per B$ e gli ultimi 4 per S$: possiamo schematizzare il tutto in questo modo:




Questa istruzione FIELD richiede che la lunghezza del buffer sia almeno di 79 (30+45+4) byte: nell'opzione /S:*b*, *b* deve perciò essere maggiore o uguale a 79 (per esempio: BASIC /S:100). Il buffer è lungo 128 byte per default e può essere al massimo di 32767 byte.
Alcuni comandi speciali sono riservati per assegnare valori alle variabili di tipo stringa che compaiono nell'istruzione FIELD: non dovete mai usare queste variabili in istruzioni di assegnamento o di input, altrimenti provochereste errori nella lettura o scrittura del file ad accesso diretto. Ad esempio, se aveste eseguito la precedente istruzione FIELD, la seguente istruzione sarebbe errata:

```
420 INPUT A$
```

Uno stesso file può essere definito per mezzo di diverse istruzioni FIELD: supponete infatti che il file A:ANAGRAFE.DAT ad accesso diretto venga aperto come file numero 8 con lunghezza del record di 20 caratteri e che vengano poi eseguite le istruzioni:

```
        .
        .
        .
200 FIELD #8, 10 AS COGNOME$, 6 AS NOME$, 2 AS ETA$,
2 AS REDDITO$
        .
        .
        .
        .
340 FIELD #8, 16 AS COGNOMENOME$, 4 AS NULLA$
        .
        .
        .
```

Più tardi, recuperiamo un record dal file: questi 20 byte potrebbero contenere questi caratteri:

| ToppinelliMarina2820 |
|---|

Se ora usiamo la variabile COGNOME$ otterremo la stringa "Toppinelli", mentre con la variabile COGNOMENOME$ otterremo "ToppinelliMarina".

## COME SCRIVERE DATI NEL BUFFER

Dopo aver stabilito il formato del record, potete usare le istruzioni LSET e RSET per disporre i dati nei campi a loro riservati nel buffer. La sintassi di queste istruzioni è:

LSET *varstr*=X$
RSET *varstr*=X$

Il parametro *varstr* indica una variabile di tipo stringa definita in una precedente istruzione FIELD, mentre X$ è l'espressione di tipo stringa da inviare al buffer. Se la variabile X$ è più corta della lunghezza riservata, l'istruzione LSET giustifica a sinistra la stringa (riempiendo di spazi vuoti lo spazio rimasto) nel buffer, mentre RSET la giustifica a destra, sempre riempiendo lo spazio in eccesso con spazi bianchi. Se X$ è più lunga dello spazio riservato, vengono in ogni caso tagliati i caratteri in eccesso a destra.

### Conversione di numeri in stringhe

Come avrete notato, i file ad accesso diretto memorizzano solo dati di tipo stringa, perciò, qualora sia necessario inserirvi dati numerici, dovete dapprima trasformarli in stringhe: per questa operazione avete a disposizione tre funzioni:

X$=MKI$(*espressione intera*)
X$=MKS$(*espressione in precisione semplice*)
X$=MKD$(*espressione in doppia precisione*)

Come potete vedere, ogni tipo di numero ha la propria funzione di conversione e ciò perché ogni precisione numerica richiede una diversa quantità di memoria per contenere la versione in formato stringa del numero. La funzione MKI$ converte un intero in una stringa di due byte, MKS$ un numero in precisione semplice in una stringa di quattro byte, MKD$ un numero in doppia precisione in una stringa di otto byte.

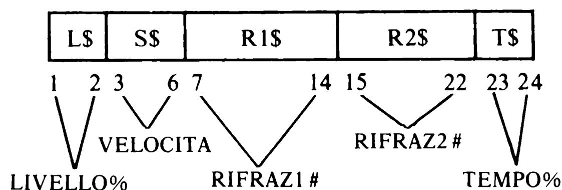
La lunghezza delle stringhe che rappresentano numeri è molto importante nel momento in cui dovete riservare lo spazio all'interno del record con un'istruzione FIELD; supponete, infatti, di voler memorizzare cinque numeri in un record: due interi, uno in precisione semplice ed altri due in doppia precisione. Un'istruzione corretta potrebbe essere:

```
3500 FIELD #5, 2 AS L$, 4 AS S$, 8 AS R1$,
     8 AS R2$, 2 AS T$
```

Dopo che il programma ha generato i valori da memorizzare, questa sequenza di istruzioni invierà le variabili numeriche nel buffer del file:

```
4000 LSET L$=MKI$(LIVELLO%)
4100 LSET S$=MKS$(VELOCITA)
4200 LSET R1$=MKD$(RIFRAZ1#)
4300 LSET R2$=MKD$(RIFRAZ2#)
4400 LSET T$=MKI$(TEMPO%)
```

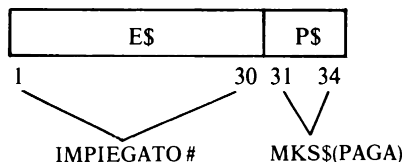
Il buffer verrà riempito in questo modo:




L'istruzione seguente, invece, predispone il record a contenere il nome di un impiegato e la sua retribuzione:

```
200 OPEN "A:PAGHE.DAT" AS #3 LEN=34
210 FIELD #3, 30 AS E$, 4 AS P$
     .
     .
     .
290 LSET E$=IMPIEGATO$
300 LSET P$=MKS$(PAGA)
     .
     .
```

Dopo l'esecuzione delle linee 290 e 300, il buffer conterrà:




## COME SCRIVERE I RECORD NEL FILE

Quando un record è stato completato (cioè avete riempito il buffer assegnando valori a tutte le variabili di tipo stringa contenute nel record) viene scritto nel file per mezzo dell'istruzione PUT, la cui sintassi è:

PUT [#] *numfile* [,*numero record*]

Il parametro *numfile* è sempre il numero del file specificato nell'istruzione OPEN; se non viene indicato il *numero record*, PUT pone il record nella prima locazione disponibile, come in un file sequenziale, altrimenti, scegliete il punto in cui sistemare il record all'interno del file per mezzo dell'opzione *numero record*, che è un intero variabile tra 1 e 16777215. Se ad esempio il file A:PAGHE.DAT è stato aperto come file numero 3, come nel caso precedente, la seguente istruzione porrà il nuovo record nella decima locazione del file:

```
340 PUT #3, 10
```

Possiamo esemplificare meglio l'azione di questa istruzione con il seguente schema:

File A:PAGHE.DAT

| **Record #** | **Dati** |
|---|---|
| 1 | Bruno 1200 |
| 2 | Marco 1150 |
| 3 | Roberto 1400 |
| : | : |
| 10 | Giorgio 900 ← Buffer |
| : | : |
| 300 | PierPaolo 1100 |

Buffer: | Giorgio | 900 |

Se non specificate il numero del record nella successiva istruzione PUT, il prossimo record sarà il numero 11.

## COME LEGGERE I RECORD DAL FILE

Per recuperare i record di un file ad accesso diretto usate l'istruzione GET, il cui formato è:

GET [#] *numfile* [,*numero record*]

Il parametro *numfile* è lo stesso presente nell'istruzione PUT, mentre il *numero record* può essere indicato quando si voglia che un particolare record del file sia trasferito nel buffer.
La seguente istruzione, perciò, serve per leggere il 274-esimo record:

```
540 GET #3, 274
```

e la sua azione può essere così illustrata:

File A:PAGHE.DAT

| **Record #** | **Dati** |
|---|---|
| 1 | Bruno 1200 |
| 2 | Marco 1150 |
| 3 | Roberto 1400 |
| : | : |
| 274 | Daniele 980 ← |
| : | : |
| 300 | PierPaolo 1100 |

Buffer

| Daniele | 980 |
|---|---|

Se non indicate il *numero record*, invece, otterrete semplicemente il record che segue immediatamente l'ultimo record letto: la seguente istruzione ora leggerà il 255-esimo record dello stesso file:

```
590 GET #3
```

Se cercate di recuperare un record che non sia stato scritto in precedenza, possono accadere due cose: se il *numero record* da voi indicato è minore dell'ultimo record scritto nel file, non è possibile sapere a priori quale record venga letto, mentre se il numero indicato è maggiore di tutti quelli presenti, il buffer sarà riempito con la stringa nulla. Se il *numero record*, sia nell'istruzione GET che in PUT è minore di 1 o maggiore di 16777215, si incorre nell'errore BAD RECORD NUMBER.

## COME LEGGERE I DATI DAL BUFFER

Per mezzo dell'istruzione GET avete caricato nel buffer un record dal file prescelto ed ora potete leggere i singoli dati sia utilizzando di nuovo le

variabili di tipo stringa definite nell'istruzione FIELD per quel buffer, sia trattando il buffer come un file sequenziale e leggendo un carattere per volta con le istruzioni INPUT# e LINE INPUT#.
Per raggiungere un determinato termine nel record, usate le variabili di tipo stringa; supponiamo che il buffer contenga i seguenti dati:

| | |
|---|---|
| Gatto | 1800 |

Poiché il buffer era stato definito con l'istruzione:

```
210 FIELD #3, 30 AS E$, 4 AS P$
```

potete leggere il nome presente nel buffer con l'istruzione:

```
NOMECORRENTE$ = E$
```

che assegna il nome Gatto alla variabile di tipo stringa di nome NOMECORRENTE$.

### Come convertire le stringhe numeriche in numeri

Se uno dei termini contenuti in un record rappresenta un numero, dovete usare le funzioni CVI, CVS e CVD per riconvertire la stringa in numero. Queste sono naturalmente le funzioni opposte a MKI$, MKS$ e MKD$ e la loro sintassi è:

X=CVI (*stringa di 2 byte*)
X=CVS (*stringa di 4 byte*)
X=CVD (*stringa di 8 byte*)

La precisione della variabile numerica X deve concordare con la precisione (intera, semplice, doppia) della funzione usata; anche la lunghezza della stringa deve combinare con il tipo della funzione: due byte per CVI, quattro byte per CVS e otto byte per CVD. Utilizzando sempre il buffer dell'esempio precedente, che appare in questo modo

| | | |
|---|---|---|
| Buffer | Gatto | 1800 |

ricordiamo che il campo numerico era stato definito nella relativa istruzione FIELD come P$; per leggere il numero del buffer, perciò, usiamo questa istruzione:

```
610 VALORE=CVS(P$)
```

che assegna il valore 15.32 alla variabile VALORE.
Nella Tabella 7.2 sono riassunte le corrispondenze tra le funzioni di conversione numero/stringa.

**Tabella 7.2** Funzioni di conversione numero/stringa

| | **Da numero a stringa** | **Da stringa a numero** |
|---|---|---|
| Intero | A$ = MKI$(Z%) | Z% = CVI(A$) |
| Precisione semplice | B$ = MKS$(Y!) | Y! = CVS(B$) |
| Doppia precisione | C$ = MKD$(X#) | X# = CVD(C$) |

## COME CONOSCERE LO STATO DI UN FILE

Come per i file sequenziali, alcune funzioni BASIC possono darvi indicazioni riguardanti lo stato di un file ad accesso diretto; queste sono le funzioni LOC e LOF. La funzione EOF, usata per i file sequenziali, qui non ha alcun effetto.
Le due funzioni si comportano con i file ad accesso diretto esattamente come con i file sequenziali. L'istruzione:

```
340 POSIZIONE=LOC(6)
```

in un file ad accesso diretto assegna alla variabile POSIZIONE il numero dell'ultimo record letto (o scritto) nel file numero 6.
Nello stesso modo, pensando che il file numero 1 sia ad accesso diretto, l'istruzione:

```
500 LUNGHEZZAFILE1=LOF(1)
```

assegna alla variabile LUNGHEZZAFILE1 il numero di caratteri (byte) contenuti nel file numero 1.
Pensiamo che il file prima definito, A:PAGHE.DAT, contenga 273 record, ciascuno lungo 34 byte: l'effettiva lunghezza del file è così di 9282 byte. L'istruzione:

```
1010 NUMEROBYTE=LOF(3)
```

assegnerà perciò alla variabile NUMEROBYTE il valore 9282.
Per conoscere il numero di record di un file ad accesso diretto, dovete compiere i passi seguenti:

1. Ricavare la lunghezza del record dall'istruzione OPEN
2. Trovare il numero di byte contenuti nel file per mezzo della funzione LOF
3. Dividere il numero di byte per la lunghezza del record

Come già visto, il risultato sarà il numero di record che compongono il file.

## UN ESEMPIO DELL'USO DEI FILE AD ACCESSO DIRETTO

Nella Figura 7.3 riportiamo un breve programma di gestione di un database con l'ausilio di file ad accesso diretto; il programma crea un file nel quale ogni record è composto da: un nome, un numero di conto e l'ammontare del conto.

```
10  '***ESEMPIO DI FILE AD ACCESSO DIRETTO***
20  '
30  '  Questo programma viene utilizzato per creare ed
 aggiornare
40  ' un elenco di clienti, per un massimo di 100 nomi
, che contenga
50  ' il nome, il numero del conto e l'ammontare
60  ' per ciascuno.
70  'Il file viene memorizzato sul disco inserito nel
drive A:
80  ' Questo programma richiede il Disk BASIC o l'Adva
nced BASIC.
90  '
100 '
110 ' ****CORPO PRINCIPALE DEL PROGRAMMA****
120 '
130 '
140 GOSUB 230  'Inizializzazione
150 GOSUB 360  'Menu del programma
160 ON SCELTA% GOSUB 520,930,1270,1590,1880 'Esegue le
 subroutine in accordo con la scelta fatta
170 GOTO 150
200 '***SUBROUTINE PER LA PREDISPOSIZIONE DI
205 '   VARIABILI,SCHERMO, DISCO E FILE ***
210 '
220 '
230 DIM AMMONT(100), PFLAG(100)
240 CLS: KEY OFF
```

(continua)

```
250 PRINT "INSERIRE IL DISCO CON L'ELENCO DEI CLIENTI
NEL DRIVE A"
260 PRINT "PREMERE UN TASTO QUALSIASI PER RIPARTIRE"
270 IF INKEY$="" THEN 270
280 OPEN "A:CLIENTI.LST" AS #1 LEN=27
290 FIELD #1, 1 AS FLAG$, 20 AS NOMECLIENTE$, 2 AS CON
TO$, 4 AS AMMONTARE$
300 RETURN
310 '
320 '
330 '***SUBROUTINE PER LA VISUALIZZAZIONE DEL MENU E
335 '    LA SCELTA DELL'OPERATORE***
340 '
350 '
360 CLS
370 PRINT "   PROGRAMMA DI GESTIONE DELL'ELENCO DEI CL
IENTI"
380 PRINT
390 PRINT
400 PRINT "OPZIONI CONSENTITE DAL PROGRAMMA -"
410 PRINT
420 PRINT "1. AGGIORNARE O CANCELLARE DATI DI UN SINGO
LO CLIENTE"
430 PRINT "2. ORDINARE L'ELENCO SECONDO L'AMMONTARE DE
L CONTO"
440 PRINT "3. STAMPARE L'ELENCO DEI CLIENTI"
450 PRINT "4. AGGIUNGERE UN NUOVO CLIENTE"
460 PRINT "5. TERMINARE IL PROGRAMMA E MEMORIZZARE IL
FILE AGGIORNATO"
470 LOCATE 20,1
480 INPUT "INSERITE LA VOSTRA SCELTA (1-5) ",SCELTA%
490 IF SCELTA% <1 OR SCELTA% >5 THEN GOTO 500 ELSE RET
URN 'Controllo dell'input
500 PRINT "PER FAVORE, INSERITE UN NUMERO COMPRESO TRA
 1 E 5. PREMERE UN TASTO QUALSIASI PER CONTINUARE.":
FOR X=0 TO 500: NEXT X
510 GOTO 360
520 '
530 '
540 ' **** SUBROUTINE PER AGGIORNARE O CANCELLARE I
545 '       DATI DI UN CLIENTE ****
550 '
560 '
570 CLS
580 T$="F":CLIENTECORR$=""
590 X%=1
600 INPUT "VOLETE INDICARE IL NUMERO DEL CONTO (S/N) ?
",A$
610 IF A$="N" THEN 650
```

(continua)

```
620 T$="S":INPUT "INSERIRE IL NUMERO DI CONTO - ",I%
630 IF I%=0 THEN PRINT "IL NUMERO DI CONTO DEVE ESSERE
 MAGGIORE DI 0 - PREMERE UN TASTO QUALSIASI PER CONTIN
UARE" ELSE 660
640 IF INKEY$="" THEN 640 ELSE 570
650 T$="N":INPUT "INSERIRE IL NOME DEL CLIENTE - ", CL
IENTECORR$
660 X%=1
670 GET #1,X%
680 IF T$="S" AND I%=CVI(CONTO$) THEN 700
690 IF T$="N" AND LEFT$ (NOMECLIENTE$,LEN(CLIENTECORR$
))=CLIENTECORR$ THEN 700 ELSE 710
700 IF FLAG$="D" THEN 910 ELSE 750  ' FLAG$ indica se
il record contenga termini  cancellati
710 IF X%<LOF(1)/27 THEN X%=X%+1: GOTO 670 ELSE 910 '
Continua la ricerca
720 '
730 ' Visualizza il cliente immesso
740 '
750 CLS
760 PRINT
770 PRINT "NOME                    "NOMECLIENTE$
780 PRINT
790 PRINT "NUMERO CONTO            "CVI(CONTO$)
800 PRINT
810 PRINT "AMMONTARE CONTO         ";
820 PRINT USING "£########"; CVS(AMMONTARE$)
830 PRINT
840 PRINT "SE VOLETE MODIFICARE L'AMMONTARE DEL CONTO,
 INSERITE LA NUOVA CIFRA"
850 INPUT "ALTRIMENTI PREMETE ENTER - ", NUOVOAMM
860 IF NUOVOAMM=0 THEN 870 ELSE LSET AMMONTARE$=MKS$(N
UOVOAMM):GOTO 890
870 INPUT "VOLETE CANCELLARE QUESTO CLIENTE (S/N) ?",
A$
880 IF A$ <> "S" THEN RETURN ELSE LSET FLAG$="D":LSET
NOMECLIENTE$=SPACE$(20)
890 PUT #1,X%
900 RETURN
910 INPUT "AVETE INSERITO UN NOME O UN NUMERO DI CONTO
 ERRATO -  BATTETE 'T' PER TERMINARE, UN ALTRO QUALSIA
SI TASTO PER RIPROVARE ",A$
920 IF A$="T" THEN RETURN ELSE 520
930 '
940 '
950 '  ****VISUALIZZA L'ELENCO DEI CLIENTI SECONDO
955 '      L'AMMONTARE DEL CONTO****
960 '
970 '
```

(continua)

```
980 CLS
990 PRINT "    - CLIENTE -                    - NUMERO CONT
O -           - AMMONTARE CONTO-"
1000 PRINT
1010 CONTARECORD=0:RECCORRENTE=0
1020 FOR X=1 TO LOF(1)/27
1030 GET #1,X
1040 IF "D"=FLAG$ THEN PFLAG(X)=1:CONTARECORD=CONTAREC
ORD+1:GOTO 1060 ELSE PFLAG(X)=0 ' Se il record è stato
 cancellato, definisce PFLAG() in modo da
1041 'eliminare il record dall'ordinamento; incrementa
 CONTARECORD
1050 AMMONT(X)=CVS(AMMONTARE$) 'Riempie il vettore dei
 conti per ordinarlo
1060 NEXT X
1070 TEST=0
1080  FOR X=1 TO LOF(1)/27  ' Cerca il conto piu' alto
 non ancora visualizzato
1090 IF PFLAG(X) THEN 1130
1100 IF AMMONT(X) < TEST THEN 1130
1110 TEST = AMMONT(X)
1120 RECCORRENTE=X
1130 NEXT X
1140 IF RECCORRENTE=0 THEN PRINT "L'ELENCO DEI CLIENTI
 E' VUOTO. PREMERE UN TASTO QUALSIASI PER RITORNARE AL
 MENU PRINCIPALE.": GOTO 1250
1150 IF PFLAG(RECCORRENTE)=1 THEN 1240 ' Controlla se
il record trovato non sia gia' stato visualizzato
1160 PFLAG(RECCORRENTE)=1  ' Definisce il flag indican
te che il record viene visualizzato, per indicare che
e' gia' stato elencato
1180 GET #1, RECCORRENTE
1190 PRINT NOMECLIENTE$ TAB(33) CVI(CONTO$) TAB(62); '
Elenco dei clienti
1200 PRINT USING "£########"; CVS(AMMONTARE$)
1210 CONTARECORD=CONTARECORD+1
1220 IF CONTARECORD=LOF(1)/27 THEN 1240 ' Controlla ch
e tutti i record siano stati visualizzati
1230 GOTO 1070  ' Continua la ricerca
1240 PRINT :PRINT "L'ELENCO DEI CLIENTI E' COMPLETO.
PREMERE UN TASTO QUALSIASI PER RITORNARE AL MENU PRINC
IPALE"
1250 IF INKEY$="" THEN 1250 ELSE RETURN
1260 '
1270 '
1280 ' ***SUBROUTINE DI STAMPA ELENCO DEI CLIENTI***
1290 '
1300 '
1310 ON ERROR GOTO 1510
```

(continua)

```
1320 LPRINT  ' Controlla che la stampante funzioni
1330 INPUT "INSERIRE LA DATA ODIERNA (GIORNO-MESE-ANNO
) - ", OGGI$
1340 LPRINT CHR$(12) ' Invia un carattere di cambio-pa
gina alla stampante
1350 LPRINT CHR$(14) TAB(3) "ELENCO DEI CLIENTI AL "OG
GI$ ' Stampa l'intestazione
1360 LPRINT
1370 LPRINT
1380 LPRINT "    - CLIENTE -           - NUMERO CONTO -
        - AMMONTARE CONTO -"
1390 FOR X=1 TO LOF(1)/27
1400 GET #1,X
1410 IF FLAG$="D" THEN 1450
1420 LPRINT
1430 LPRINT NOMECLIENTE$ TAB(33) CVI(CONTO$) TAB(53);
1440 LPRINT USING "£########"; CVS(AMMONTARE$)
1450 NEXT X
1460 LPRINT CHR$(12) CHR$(12) ' Invia due caratteri di
 cambio-pagina
1470 RETURN
1480 '
1490 ' Routine per l'errore sulla stampante
1501 ' causato dal ritardo del cambio-pagina
1502 '
1510 IF ERL=1340 OR ERL=1460 THEN RESUME NEXT '  Inibi
sce il messaggio d'errore
1520 IF ERR=27 THEN PRINT "PREGO, CONTROLLARE LA STAMP
ANTE":PRINT
1530 FOR F=400 TO 1000 STEP 100
1540 SOUND F,.5: SOUND 32767, .5
1550 NEXT F
1560 RESUME
1570 '
1580 '
1590 '***SUBROUTINE DI AGGIUNTA NUOVO CLIENTE****
1600 '
1610 '
1620 X = 1
1630 GET #1, X
1634 IF FLAG$="G" THEN X=X+1 :GOTO 1630
1640 IF FLAG$="D" THEN 1650
1650 RECORD=X
1660 CLS
1670 INPUT "INSERIRE IL NUOVO NOME - ",CLIENTE$
1680 PRINT
1690 INPUT "INSERIRE IL NUMERO DEL CONTO - ",CONTO%
1700 PRINT
1710 IF CONTO%=0 THEN PRINT "RIINSERIRE IL NUMERO, MAG
GIORE DI 0":GOTO 1690
```

(continua)

```
1720 INPUT "INSERIRE L'AMMONTARE DEL CONTO - ", AMMONT
ARE
1730 PRINT :PRINT "IL NUOVO INSERIMENTO E': "
1740 PRINT "CONTO NUMERO: " CONTO% " NOME: ";
1750 PRINT USING "\                  \";CLIENTE$;
1760 PRINT "  AMMONTARE: ";
1770 PRINT USING "£########";AMMONTARE
1780 PRINT :INPUT "L'INSERIMENTO E' CORRETTO (S/N) ";
A$
1790 IF A$="N" THEN 1660
1800 LSET NOMECLIENTE$=CLIENTE$
1810 LSET CONTO$=MKI$(CONTO%)
1820 LSET AMMONTARE$=MKS$(AMMONTARE)
1830 LSET FLAG$="G"
1840 PUT #1, RECORD
1850 RETURN
1860 '
1870 '
1880 ' ****MEMORIZZA IL FILE AGGIORNATO E TERMINA
1885 '     IL PROGRAMMA****
1890 '
1900 '
1910 CLOSE
1920 CLS
1930 ON ERROR GOTO 0
1940 END
```

**Figura 7.3** Un esempio di file ad accesso diretto

Il corpo principale del programma inizia con la chiamata ad una subroutine di inizializzazione che parte dalla linea 230: qui vengono dimensionati due array che saranno utilizzati più tardi per una routine di ordinamento. Viene preparato lo schermo e vi è richiesto di inserire nel drive A: il disco contenente il file di dati: se il disco non contiene il file, questo sarà creato dal programma stesso. In ogni caso il disco deve almeno essere stato formattato dal DOS prima di questo impiego.
La routine di inizializzazione definisce il file A:CLIENTI.LST come file numero 1; l'istruzione FIELD della linea 290 riserva lo spazio per una stringa di un solo carattere (flag) che verrà usata per indicare se un determinato record è stato scritto o cancellato, lo spazio per il nome del cliente, lungo 20 caratteri, lo spazio per il numero (intero) del conto, di due byte e lo spazio per l'ammontare del conto, di quattro byte per contenere un numero in precisione semplice.
La routine successiva, che inizia nella linea 360, visualizza il menu del programma, cioè un elenco di opzioni possibili, seguite dalla richiesta di un input da parte dell'operatore; se l'input non è valido, viene visualizza-

to un messaggio (linea 500) e l'input viene ripetuto.
Se scegliete l'opzione numero 1, viene eseguita la routine di aggiornamento o cancellazione del dato relativo al cliente, che vi richiede il numero del conto del cliente (linea 620) o il nome del cliente (linea 650) e controlla lungo tutto il file se esiste un tale nome (linea 690) o numero di conto (linea 680).
La routine che controlla il nome del cliente ricerca una corrispondenza con gli *n* caratteri più a sinistra della variabile NOMECLIENTE$ contenuta nel record, ove *n* è la lunghezza del nome inserito, cioè CLIENTECORR$: questo assicura che gli spazi che eventualmente seguono un nome non impediscano l'esatto riconoscimento.
Se viene trovata una possibile corrispondenza, il programma controlla la variabile FLAG$: se il record è stato cancellato, questa è uguale a D ed il riconoscimento non è valido, altrimenti il record viene visualizzato; la routine di visualizzazione inizia nella linea 750.
Se non avvengono riconoscimenti, viene visualizzato un messaggio (linea 910) e viene presentata la scelta tra provare con un altro nome o numero oppure ritornare al menu iniziale.
Dopo aver trovato e visualizzato un record, le linee dalla 840 alla 860 vi permettono di cambiare l'importo; in modo analogo le linee 870 e 880 vi permettono di cancellare uno dei termini ed in questo caso la variabile FLAG$ relativa viene posta uguale a D. La routine termina con l'istruzione PUT, nella linea 890, che scrive nel file la nuova versione del record.
Le opzioni del menu dalla 2 alla 5 permettono di visualizzare l'elenco dei clienti ordinato secondo l'ammontare del conto, stampare questo elenco, aggiungere un nuovo cliente e terminare il programma. Un campione dell'output ottenuto con l'opzione 3 è riportato nella Figura 7.4.

ELENCO DEI CLIENTI AL 11-02-1985

| - CLIENTE - | - NUMERO CONTO - | - AMMONTARE CONTO - |
|---|---|---|
| Ferdinando Buio | 33 | £ 120500 |
| Franco Franchi | 1 | £ 340780 |
| Fosca Foscarini | 45 | £ 1000000 |
| Franco M. Fila | 5 | £ 795000 |
| Fatima L. Occhi | 2 | £ 983000 |

**Figura 7.4** Esempio di stampa dell'elenco clienti

## 7.4 File pseudo-sequenziali

Un file pseudo-sequenziale è un file ad accesso diretto a cui si accede, però, come se fosse un file sequenziale: combina perciò l'efficienza dei file ad accesso diretto con il modo sequenziale di gestire input ed output. Perciò, se i dati devono entrare ed uscire dal programma in modo sequenziale suddivisi in grandi blocchi, risulta più vantaggioso immagazzinarli in un file pseudo-sequenziale.

### COME ACCEDERE AD UN FILE PSEUDO-SEQUENZIALE

Ci sono due punti a cui bisogna prestare attenzione nell'accedere ad un file pseudo-sequenziale; il primo è che potete accedere al buffer del file ad accesso diretto in modo sequenziale usando le istruzioni del tipo PRINT#, LINE INPUT# e così via.
Il secondo è che viene generato un errore di overflow del campo (FIELD OVERFLOW) ogni volta che il programma cerca di scrivere o leggere troppi dati nel buffer; il trasferimento dei dati tra il buffer ed il file pseudo-sequenziale avviene grazie al rilevamento di questo errore.
Se volete usare un file pseudo-sequenziale, dovete innanzitutto predisporre all'uso il file con la versione dell'istruzione OPEN riservata ai file ad accesso diretto; dal momento però che accederete al buffer in modo sequenziale, non è necessaria l'istruzione FIELD. A questo punto dovete preparare l'intercettamento dell'errore di overflow del campo.
Il passo successivo è la lettura o la scrittura dei dati nel buffer usando le istruzioni riservate ai file sequenziali: potete considerare il buffer come un pozzo senza fondo di dati. Però, quando il buffer si riempie (in output) o si svuota completamente (in input), si verifica l'errore di overflow del campo. Il compito della subroutine di intercettamento è di eseguire l'azione richiesta: se il buffer che state scrivendo è pieno, la routine deve trasferire (PUT) il contenuto del buffer nel file pseudo-sequenziale, mentre se il buffer è vuoto, in input, deve recuperare (GET) un record dal file pseudo-sequenziale.
Questo è il nucleo del processo: il buffer viene scritto e letto come nell'accesso sequenziale e quando è del tutto pieno o vuoto si genera un errore e la relativa routine di intercettamento scrive il buffer nel file o legge un record dal file nel buffer.

## COME APRIRE UN FILE PSEUDO-SEQUENZIALE

La prima operazione da compiere per usare un file pseudo-sequenziale (PS) è di aprirlo come file ad accesso diretto con record di lunghezza uguale a 512 byte: questa lunghezza è ottimale per diminuire il tempo impiegato a trasferire i dati da o su disco, in quanto il buffer non si riempie o si svuota così rapidamente come quando la lunghezza del record è di 128. Ad esempio:

```
340 OPEN "PSEUFILE" AS #1 LEN=512
```

## COME SCRIVERE DATI NEL BUFFER

Ogniqualvolta un dato è pronto per essere scritto in un file PS, utilizzate l'istruzione PRINT#: questa serve per immettere i dati di tipo stringa nel buffer del file ad accesso diretto; l'istruzione PRINT# USING invece viene usata quando è necessario dare un formato particolare ai dati.
Quando viene eseguita l'istruzione PRINT #, il buffer inizia a riempirsi, fino al riempimento completo, quando interviene la routine di intercettamento. Ecco le istruzioni che predispongono l'intercettamento per file pseudo-sequenziali:

```
350 ON ERROR GOTO 1000
      .
      .
500 PRINT #1, DATI$
      .
      .
      .
1000 IF ERR = 50 THEN 1200
      .
      .
      .
1200 PUT #1 : RESUME
```

In questo esempio, l'errore a cui si riferisce la linea 350 si verifica quando la linea 500 cerca di scrivere in un buffer già pieno; a questo punto viene eseguita la routine di intercettamento che inizia nella linea 1000: questa esegue l'istruzione PUT#1 che vuota il buffer e scrive il record nel file ad accesso diretto #1, cioè il PSEUFILE definito prima. Osservate che i record vengono scritti in ordine sequenziale, poiché nessun numero di record viene inserito nell'istruzione PUT#1.
Rimane ora un unico problema e cioè come segnalare la fine del file in

modo che questo venga poi letto correttamente: la fine del file dev'essere indicata con un carattere che sia identificato univocamente in input; potremmo scrivere un carattere EOF nel file, appena prima di concluderlo:

```
600                  'Il file di output e' completo
610 PRINT #1,"EOF"  'Pone un carattere EOF nel
buffer
620 ON ERROR GOTO 0 'Disabilita l'intercettamento
poiche' il file e' completo
630 PUT #1 : CLOSE  'Libera il buffer e chiude il
file
```

Osservate che in questo caso il carattere EOF non è stato scelto per qualche motivo intrinseco: ogni altro carattere avrebbe potuto svolgere lo stesso ruolo purché fosse identificabile senza ambiguità in input come fine del file.

## COME LEGGERE DATI DAL BUFFER

Per recuperare i dati da un file PS, questo dapprima deve essere aperto come file ad accesso diretto:

```
100 OPEN "PSEUFILE" AS #1 LEN=512
```

e quindi vi si deve accedere per mezzo del buffer dei file ad accesso diretto, con l'istruzione INPUT# o LINE INPUT#:

```
200 INPUT#1, DATI$
```

Un'istruzione di questo tipo legge il primo termine che si trova nel buffer; notate che i vari termini devono essere separati da una virgola, un ritorno carrello o a capo per essere riconosciuti separatamente. Come prima, potete continuare a leggere il buffer in modo sequenziale.
Ogni volta che il BASIC cerca di leggere un buffer vuoto con un'istruzione INPUT# o LINE INPUT#, si verifica un errore di overflow del campo e a questo punto l'opportuna routine di intercettamento deve eseguire un GET# per leggere il record successivo e porlo nel buffer.

```
110 ON ERROR GOTO 2000
      .
      .
      .
```

```
2000 IF ERR=50 THEN 2200
     .
     .
     .
2200 GET #1 : RESUME
```

Per concludere il processo di input, dobbiamo cercare il segnale che avevamo posto alla fine del file: si potrebbe fare questo dopo ogni istruzione che legge il buffer. Eccone un esempio:

```
200 INPUT#1, DATI$
210 IF DATI$="EOF" THEN ON ERROR GOTO 0 : CLOSE
```

## UN ESEMPIO DI FILE PSEUDO-SEQUENZIALE

Le Figure 7.5 e 7.6 mostrano come si possa creare, copiare e visualizzare un file pseudo-sequenziale. Il primo programma crea il file A:PSEUFILE.DAT e lo riempie con tre stringhe: ogni stringa di caratteri è ripetuta per 100 volte nel file. Il secondo programma copia i dati del file A:PSEUFILE.DAT in un secondo file, pseudo-sequenziale, di nome A:ALTROFIL.DAT; i dati copiati vengono anche visualizzati sullo schermo.

```
10 '** ESEMPIO NUMERO 1 DI FILE PSEUDO-SEQUENZIALE **
20 '
30 '  Questo programma crea un file pseudo-sequenziale
 di nome
40 ' "PSEUFILE" residente sul disco inserito nel drive
 A.
50 ' Il file contiene tre stringhe di caratteri ripetu
te ciascuna
60 ' 100 volte.
70 ' Osservate che questo programma richiede che il Di
sk o
80 ' l'Advanced BASIC sia stato avviato con
90 ' l'opzione /S:512 , cioe' con un buffer di lunghez
za 512. Inoltre dovete
100 ' inserire un disco gia' formattato nel drive A pr
ima di iniziare .
110 '
120 CLS
130 PRINT "INSERITE IL DISCO NEL DRIVE A. PREMETE UN T
ASTO PER CONTINUARE."
140 IF INKEY$="" THEN 140
150 OPEN "A:PSEUFILE.DAT" AS #1 LEN=512
```

(continua)

```
160 ON ERROR GOTO 300    '  Abilita l'intercettamento de
gli errori
161 DATA "Primo","Secondo","Terzo"
162 FOR E=1 TO 3
163 READ ENTRY$
170 FOR A=1 TO 100
180 PRINT #1, ENTRY$ 'Inserisce la stringa nel buffer
del file
190 NEXT A
191 NEXT E
200 '
210 'Termina il file
220 '
230 PRINT #1, "//EOF" ' Pone un carattere end-of-file
al termine di PSEUFILE
240 ON ERROR GOTO 0
250 PUT #1: CLOSE ' Elimina ogni carattere rimasto nel
 buffer
260 END
270 '
280 ' Routine di intercettamento
290 '
300 PUT #1 :RESUME ' Quando il buffer e' troppo pieno,
 lo vuota e riprende l'input da tastiera
```

**Figura 7.5** Un esempio di come creare un file pseudo-sequenziale

```
10 '** ESEMPIO NUMERO 2 DI FILE PSEUDO-SEQUENZIALE **
20 '
30 '  Questo programma legge le informazioni memorizza
te nel file pseudo-sequenziale
40 ' PSEUFILE e le trasferisce in un altro file pseudo
-sequenziale
50 ' di nome ALTROFIL.Inoltre, ogni termine viene  vis
ualizzato sullo schermo
60 ' durante il trasferimento. Anche questo programma
richiede che il Disk o
70 ' l'Advanced BASIC siano avviati con l'opzione /S:
512.
80 '
90 PRINT "INSERITE NEL DRIVE A IL DISCO CON IL FILE
'PSEUFILE'."
100 PRINT "PREMETE UN TASTO QUALSIASI PER CONTINUARE"
110 IF INKEY$="" THEN 110
120 OPEN "A:PSEUFILE.DAT" AS 1 LEN = 512   ' Apre i due
 file pseudo-sequenziali
130 OPEN "A:ALTROFIL.DAT" AS 2 LEN = 512
140 ON ERROR GOTO  300
```

(continua)

```
150 GET #1 ' Riempie il buffer del file numero 1
160 INPUT #1, ENTRY$  ' Prende la stringa dal buffer
170 PRINT #2, ENTRY$  ' Mette la stringa di PSEUFILE n
el buffer di ALTROFIL
180 IF ENTRY$="//EOF" THEN 240 ' Controllo sul caratte
re end-of-file
190 PRINT ENTRY$; ' Visualizza la stringa
200 GOTO 160 ' Prende la stringa successiva
210 '
220 ' Tiene conto del carattere EOF
230 '
240 ON ERROR GOTO 0 ' Disabilita l'intercettamento
250 PUT #2: CLOSE   ' Vuota il buffer di output e chiu
de i file
260 END
270 '
280 'Routine d'errore
290 '
300 IF ERL=160 THEN GET #1: RESUME ELSE 310  ' Se l'er
rore si e' verificato nella linea 160
301  ' riempie il buffer di input e riprende.
310 PUT #2: RESUME   ' Altrimenti, vuota il buffer usa
to per l'output e riprende
```

**Figura 7.6** Programma per copiare e visualizzare un file pseudo-sequenziale

Capitolo

# 8 La memoria del PC-IBM

La conoscenza dell'organizzazione della memoria del PC è indispensabile per riuscire a sfruttare al meglio la memoria a disposizione. Quando inizierete a scrivere programmi sempre più complessi, vi risulterà utile poter accedere direttamente ad alcune locazioni di memoria o anche disporre di una parte di memoria al di là del massimo di 64K permesso dal BASIC. In questo capitolo, perciò, troverete informazioni sulla disposizione della memoria e sul modo di accedere ad essa, sia all'interno che all'esterno dell'area riservata per i programmi in BASIC.

## 8.1 Come è organizzata la memoria

La memoria del PC è formata da ROM (*Read-Only-Memory*, memoria a sola lettura) e RAM (*Random-Access-Memory*, memoria ad accesso diretto). Il contenuto della ROM è fisso: una volta programmato (in questo caso dall'IBM), nessuno di questi chip può essere modificato; le RAM, invece, sono memorie che possono essere sia lette che scritte dall'utente.
Tra i programmi memorizzati nelle ROM troviamo le routine DOS di input/output ed il programma interprete del Cassette BASIC; l'interprete dell'Advanced o del Disk BASIC e i programmi che voi scrivete in BASIC, invece, vengono caricati nelle RAM dal disco.
Il PC è disponibile con diverse configurazioni di memoria, cioè con diverse quantità di RAM e ROM. Normalmente nel PC sono contenuti 40 kilobyte di ROM.
Tutti i PC/XT vengono venduti con 128KB di RAM. Alcuni zoccoli (allog-

giamenti predisposti per gli integrati) nella scheda di sistema — cioè il grande circuito stampato all'interno dell'unità di sistema — offrono la possibilità di inserire ulteriori 128KB di RAM; potete inoltre aggiungere altre schede di espansione nell'unità di sistema per incrementare la quantità di RAM. Per il momento la massima quantità di RAM disponibi-

**Tabella 8.1** Mappa della memoria del PC

| Decimale | Esadecimale | Funzione | Descrizione |
|---|---|---|---|
| 0<br>.<br>.<br>128KB | 00000<br>.<br>.<br>20000 | 128–256KB di RAM nella scheda di sistema | Fornito di solito con il sistema |
| 256KB<br>.<br>.<br>.<br>.<br>.<br>624K | 40000<br>.<br>.<br>.<br>.<br>.<br>9C000 | Fino a 384KB di RAM su schede opzionali inserite nei canali di I/O | Spazio RAM che può essere usato senza un particolare software |
| 640K | A0000 | Riservato | |
| 656K<br>672K<br>688K | A4000<br>A8000<br>AC000 | | Questo spazio può essere usato sia dalle schede originali IBM che di altre ditte; le schede IBM occupano lo spazio nel modo indicato |
| 704K | B0000 | Buffer per il monitor monocromatico | |
| 720K | B4000 | | |
| 736K | B8000 | Buffer per il monitor a colori | |
| 752K | BC000 | | |
| 768K | C0000 | | |
| .<br>.<br>.<br>944K | .<br>.<br>.<br>EC000 | 192KB: area di espansione della memoria | Questo spazio richiede l'esistenza di uno speciale software per essere utilizzato |
| 960K | F0000 | Riservato | |
| 976K<br>992K<br>1008K | F4000<br>F8000<br>FC000 | 48KB di ROM di sistema | I 40K finali di questa ROM vengono forniti con il sistema. |

le è di 256KB nella scheda di sistema più 384KB nelle schede di espansione dell'unità di sistema.
Il microprocessore del PC è in grado di indirizzare fino a 1048576 byte di memoria (un megabyte, o 1MB) tra ROM e RAM. Ogni byte è identificato dal proprio indirizzo che, scritto in notazione esadecimale, può variare tra 00000 e FFFFF: l'intervallo di variazione degli indirizzi viene spesso chiamato *spazio degli indirizzi* del PC.
Lo schema della memoria del PC nella Tabella 8.1 mostra come è suddiviso lo spazio degli indirizzi.

## 8.2 Segmenti di memoria

Lo spazio degli indirizzi del PC è suddiviso in blocchi di 64KB chiamati *segmenti*; un segmento può iniziare in un punto qualsiasi della memoria, purché il suo indirizzo sia un numero divisibile per 16. In altre parole, la cifra meno significativa (cioè quella più a destra) dell'indirizzo a cui inizia il segmento, quando scritto in notazione esadecimale, dev'essere zero.
I programmi BASIC vengono scritti a partire dai segmenti con il numero più basso, dopo che sono stati caricati il DOS e l'Advanced o il Disk BASIC. L'area in cui vengono caricati i programmi viene detta *segmento dei dati* BASIC.
Una locazione specifica di memoria è identificata dall'indirizzo del segmento a cui appartiene e da un valore detto *offset*, che può variare da 0 a 65535 (in decimale) o FFFFF (in esadecimale), dato che la lunghezza del segmento è di 64KB. Inoltre, poiché i segmenti possono iniziare in un qualsiasi punto della memoria purché rispettino la limitazione appena accennata sull'indirizzo di partenza, una data locazione può corrispondere a diverse combinazioni di segmenti e offset.
Se perciò volete indirizzare la locazione numero 17408 in memoria, in riferimento al segmento che inizia dall'indirizzo 00000, questa avrà un offset di 17408 (in decimale) o 4400 (in esadecimale), mentre in riferimento al segmento che parte dall'indirizzo 1000 (esadecimale), l'offset sarà 13312 (in decimale) o 3400 (in esadecimale).
Se non viene indicato altrimenti, il segmento corrente è considerato il segmento dei dati BASIC; per poter accedere a locazioni di memoria al di fuori di questo, dovete ridefinire il segmento corrente per mezzo di un'istruzione DEF SEG, con la seguente sintassi:

DEF SEG [=*indirizzo*]

Le due parole DEF e SEG devono essere separate da uno spazio, altrimenti il BASIC le interpreterà come il nome di variabile DEFSEG.

Se l'istruzione viene eseguita senza il parametro *indirizzo*, il segmento corrente diviene il segmento dati BASIC; se invece il parametro viene specificato, l'indirizzo del segmento corrente viene dato dal parametro moltiplicato per 16.
Questo parametro può variare da 0 a 65535; osservate come, una volta moltiplicato per 16, l'intervallo per il parametro copra interamente il megabyte di memoria del PC. Un esempio di come si definisca il segmento corrente è dato dall'istruzione:

```
30 DEF SEG = &HB000
```

che fa partire il segmento corrente dall'indirizzo B0000H. Tutte le successive istruzioni che utilizzano il valore del segmento corrente per calcolare gli indirizzi delle locazioni, utilizzeranno il valore B0000H. Notate che l'indirizzo in questo esempio, B0000H, è l'inizio del buffer che controlla il monitor monocromatico.
L'istruzione DEF SEG ha effetto sulle istruzioni BASIC: PEEK, POKE, BSAVE, BLOAD, CALL e USR, mentre le altre non subiscono alcuna modifica dalla ridefinizione del segmento corrente.

## 8.3 Come accedere alla memoria dal BASIC

Sono diverse le istruzioni che vi permettono di accedere direttamente alla memoria dall'interno di un programma BASIC: la maggior parte di esse si riferisce ad un indirizzo effettivo di memoria, altre utilizzano nomi simbolici. Quando compare un indirizzo numerico, questo si intende sempre riferito ad un segmento di memoria di 64KB definito in precedenza.

### LA FUNZIONE PEEK

Se volete esaminare un particolare byte di memoria, potete usare la funzione PEEK, la cui sintassi è

*byte* = PEEK(*n*)

dove il parametro *n* è un intero compreso tra 0 e 65535. La funzione restituisce in notazione decimale il contenuto del byte che si trova all'indirizzo *n* del segmento corrente. Le istruzioni:

```
400 DEF SEG = 0
410 MEMSIZE = PEEK(&H413)
```

assegnano alla variabile MEMSIZE il contenuto della locazione numero 413H: in particolare, questa locazione contiene il numero dei blocchi di memoria da 1KB contigui installati nel sistema. (BIOS assegna il valore a questo byte quando il sistema viene avviato).

## L'ISTRUZIONE POKE

Se invece volete scrivere un valore in un particolare byte di memoria, usate l'istruzione POKE, la cui sintassi è

POKE *n,byte*

Il parametro *n* è un numero compreso tra 0 e 65535 indicante l'offset all'interno del segmento corrente in cui verrà caricato *byte*, che può variare tra 0 e 255.
Consideriamo perciò l'istruzione:

```
POKE 106,0
```

che carica il valore 0 nel 106-esimo byte del segmento dati BASIC, che contiene un flag della tastiera: il BASIC cioè controlla in questa locazione per sapere se sia stato inviato un carattere dalla tastiera. Normalmente questa locazione viene letta solamente dal BASIC, mentre in questo caso noi ne forziamo il valore per indicare che non ci sono caratteri provenienti dalla tastiera.

## LE ISTRUZIONI BSAVE E BLOAD

Le istruzioni BSAVE e BLOAD permettono di salvare o caricare un intero blocco di memoria da un file BASIC su disco o su cassetta.
La sintassi dell'istruzione BSAVE è:

BSAVE *"specfile", offset, lunghezza*

Il parametro *specfile* dev'essere formato da un cammino e da un nome di file su disco, entrambi validi; l'*offset* è un'espressione compresa tra 0 e 65535, che viene aggiunta all'indirizzo del segmento corrente per definire l'indirizzo di partenza del blocco di memoria da salvare. Il parametro *lunghezza* specifica la lunghezza, in byte, del blocco da salvare. La lunghezza del blocco può essere un numero qualsiasi compreso tra 1 e 65535 byte.

La sintassi dell'istruzione BLOAD è:

BLOAD "*specfile*" [,*offset*]

Il parametro *specfile* è il nome di un file su disco che contiene i dati che si desidera caricare nella memoria del PC. Il file dev'essere stato creato precedentemente per mezzo di un'istruzione BSAVE. Il parametro opzionale *offset* specifica dove il file sarà caricato in memoria, relativamente al segmento di memoria corrente.
BSAVE e BLOAD possono essere usate come comandi sia in modo diretto, sia all'interno di un programma BASIC. In modo diretto possono essere usate per immagazzinare e caricare subroutine in linguaggio macchina. Un tipico uso di queste istruzioni in un programma è quello di salvare e recuperare una particolare videata sullo schermo.
Come esempio di queste istruzioni, vi mostriamo come salvare una pagina di schermo di 80×25 caratteri usando la scheda Colore/Grafica.
Il primo passo è calcolare l'indirizzo del buffer dello schermo. Supponiamo di dover salvare la seconda pagina di schermo tra quattro possibili. Guardando lo schema di memoria della Tabella 8.1 potete vedere che vi sono in totale 16KB di memoria per lo schermo. Poiché vi sono quattro differenti pagine di schermo nel modo Text con dimensione 80×25, ogni pagina occupa 4KB di memoria. Così la seconda pagina viene immagazzinata in memoria tra gli indirizzi B9000H e B9FFFH (B8000H+1000H = B9000H; 4K = 1000H).
Perciò se definiamo il segmento corrente con l'istruzione:

```
DEF SEG = &HB800
```

l'offset dell'inizio del buffer è 1000H. Nello stesso modo la lunghezza del file da salvare è 1000H; le seguenti istruzioni, perciò, salveranno la seconda pagina di schermo di testo nel file A:DISP1.SCN:

```
400 DEF SEG = &HB800
410 BSAVE "A:DISP1.SCN",&H1000,&H1000
```

Successivamente, supponiamo di voler caricare questa pagina appena salvata come pagina 0; le seguenti linee di programma serviranno allo scopo:

```
600 DEF SEG = &HB800
610 BLOAD "A:DISP1.SCN"
```

Si noti che l'offset non è stato specificato, poiché l'inizio della pagina 0 di un testo di 80×25 caratteri è B8000H.

## LA FUNZIONE VARPTR

Il BASIC conserva le variabili del programma all'interno della propria memoria; le loro locazioni sono determinate quando il programma viene mandato in esecuzione e possono cambiare durante l'esecuzione stessa del programma. Dunque, ogni volta che in un programma BASIC viene creato un file, all'interno della memoria viene riservata un'area in cui mantenere le informazioni relative a questo file; quest'area viene chiamata in BASIC *blocco di controllo del file*.
Gli indirizzi delle variabili e dei blocchi di controllo dei file possono essere ottenuti per mezzo della funzione VARPTR, la cui sintassi è:

X = VARPTR (*variabile*)

oppure

Y = VARPTR (*#numfile*)

La prima forma di questa funzione restituisce l'offset dell'indirizzo del primo byte della variabile; l'offset, che sarà compreso nell'intervallo da 0 a 65535, è relativo al segmento dati BASIC (DS). La variabile deve essere già stata assegnata nel programma. Il numero totale di byte che vengono riservati per una variabile dipende dal tipo di variabile (vedi Capitolo 5); per esempio, variabili numeriche intere occupano due byte, mentre una variabile stringa occupa un byte per ogni carattere della stringa.
La seconda forma di quest'istruzione restituisce l'offset nel segmento dati BASIC del blocco di controllo per il file specificato. Il blocco di controllo del file contiene informazioni inerenti al file, come ad esempio la lunghezza del file, il numero di record, il numero del dispositivo e così via.
Per fare un esempio dell'uso di VARPTR, supponiamo di dover trovare la lunghezza dei record in un file ad accesso diretto di nome A:FILETST. Dato che la lunghezza del record è memorizzata nei byte 179-esimo e 180-esimo del blocco di controllo del file, le seguenti istruzioni consentiranno di ottenere quanto richiesto:

```
900 DEF SEG ' Il segmento corrente ritorna ad essere
             il segmento dati BASIC
910 RECLENADDR=VARPTR(#3)+179 ' Il file A:FILETST è
             stato aperto (OPEN) come file #3
920 LOWLEN=PEEK(RECLENADDR) ' Prende il byte meno
             significativo della lunghezza del record
930 HILEN=PEEK(RECLENADDR+1) ' Prende il byte più signi-
             ficativo della lunghezza del record
940 RECLEN=LOWLEN+(255*HILEN) ' Calcola la lunghezza
             del record
```

## 8.4 Come modificare lo spazio di lavoro

Parecchie istruzioni consentono di modificare lo spazio in cui risiede un programma BASIC. L'opzione /M: e l'istruzione CLEAR vi permettono di riservare spazio per cose come subroutine in linguaggio Assembler, mentre la funzione FRE vi permette un uso efficiente dell'area di memoria.

### L'OPZIONE /M:

Normalmente il BASIC usa i primi 64KB di memoria disponibili come area di lavoro per i programmi e i dati. Questo spazio può essere ridotto per mezzo dell'opzione /M: quando si lavora in Disk o Advanced BASIC. Ricordate che questo parametro fa parte dei comandi BASIC e BASICA:

BASIC[A] [*"specfile"*] [<*stdin*] [[>] >*stdout*] [/F:*nfile*] [/S:*dimbuf*]
[/C:*combuf*] [/M:[*max spaziolavoro*]
[,*max dimbloc*]] [/D]

Il parametro indicante il massimo spazio lavoro può essere un intero fino a 65535 e può essere rappresentato in notazione decimale, ottale (preceduto da &0) o esadecimale (preceduto da &H).
L'opzione /M: può essere usata quando avete bisogno di riservare una certa quantità di memoria per una subroutine in linguaggio macchina, o altro.

### L'ISTRUZIONE CLEAR E LA FUNZIONE FRE

L'istruzione CLEAR e la funzione FRE vi permettono di controllare il modo in cui è organizzata la memoria.
Come abbiamo già visto nel Capitolo 5, la sintassi dell'istruzione CLEAR è:

CLEAR [,[*byte*] [,*stack*]]

Un'istruzione CLEAR senza alcun parametro cancella tutte le variabili del programma BASIC residente; così le variabili numeriche sono azzerate e alle variabili di tipo stringa viene assegnato un valore nullo.
Il parametro *byte* specifica la massima dimensione di spazio che il programma BASIC può usare, al più 64KB; questo parametro ha la stessa esatta funzione dell'opzione /M: per i comandi BASIC e BASICA.
Come nel caso dell'opzione /M:, il comando CLEAR con il parametro *byte*

può essere usato quando viene utilizzata una subroutine in linguaggio macchina in un sistema con memoria limitata.
Il parametro *stack* specifica il massimo spazio per lo stack che il programma corrente può usare. Normalmente il BASIC riserva per lo stack 512 byte o un ottavo della memoria disponibile, a seconda di quale sia più piccolo. Alcune applicazioni possono richiedere uno stack di maggiori dimensioni: ad esempio programmi che hanno varie subroutine o loop FOR...NEXT annidati, programmi che contengono complesse istruzioni PAINT o che usano strutture ricorrenti.
Le seguenti istruzioni illustrano l'uso di CLEAR:

```
340 CLEAR , &H8000 ' Predispone il massimo spazio
per il BASIC a 32K

490 CLEAR , 32768 , 1024 ' Predispone a 32K il
massimo spazio per il BASIC e ad 1K il massimo
spazio per lo stack
```

La sintassi di FRE è

Y = FRE(x)

oppure

Y = FRE(x$)

Questa funzione restituisce il numero di byte ancora liberi nello spazio di lavoro del BASIC. Se l'argomento della funzione FRE è una stringa, essa indurrà il BASIC a fare alcune operazioni di "pulizia" nello spazio di lavoro del BASIC stesso. Questa operazione risistema l'area di memoria usata per le variabili, ricuperando lo spazio inutilizzato. Normalmente il BASIC cercherà di far pulizia automaticamente ogni volta che lo spazio non è sufficiente. Per i programmi che richiedono molta memoria, un uso frequente della funzione FRE con una stringa come argomento rende più efficiente l'uso della memoria stessa.
L'argomento della funzione FRE è fittizio, cioè l'azione della funzione non è influenzata dal valore dell'argomento e l'argomento, a sua volta, non è modificato dalla chiamata alla funzione.

## 8.5 Come accedere alle subroutine in linguaggio macchina

Se vi capitasse di dover usare programmi scritti direttamente in linguaggio macchina all'interno di un programma BASIC, potrete utilizzare o l'istruzione CALL o la funzione USR. Le subroutine in linguaggio macchina possono essere immagazzinate ovunque all'interno della memoria, sia fuori che dentro il segmento dati BASIC.
Un programma in linguaggio macchina è più efficiente dell'equivalente programma BASIC; per esempio, può occuparsi delle particolari necessità di un dispositivo periferico con molta maggiore flessibilità e velocità di quanto possa fare un programma BASIC.
La sintassi dell'istruzione CALL è:

CALL *varnum* [(*variabile* [,*variabile*]...)]

Il parametro *varnum* è una variabile numerica che contiene l'offset dell'indirizzo di partenza per una subroutine in linguaggio macchina, relativamente al segmento corrente.
I parametri opzionali *variabile* vengono passati come argomenti della subroutine; gli indirizzi di queste variabili vengono immessi nello stack del BASIC quando viene eseguita l'istruzione CALL.
Un altro modo di accedere alle subroutine in linguaggio macchina è offerto dalla funzione USR; prima di poterla usare, dovrà essere stata eseguita l'istruzione DEF USR allo scopo di inizializzare l'indirizzo per una delle dieci funzioni USR che è possibile attivare in ogni momento. La sintassi dell'istruzione DEF USR è la seguente

DEF USR [*n*] = *offset*

Il parametro *n* è un intero compreso tra 0 e 9, e quando viene omesso si assume per default lo 0. Il parametro *offset* è un intero compreso tra 0 e 65535 che specifica l'indirizzo di partenza della funzione definita relativamente al segmento corrente. La sintassi della funzione USR è:

Y = USR [*n*] (x)

L'istruzione chiama la funzione appropriata, da USR 0 a USR 9, con argomento x ed assegna il risultato alla variabile Y.

Capitolo

# La grafica 9

Se possedete una scheda Colore/Grafica ed un monitor a colori, potete scrivere programmi in BASIC con cui realizzare grafici e diagrammi, disegnare linee, figure, cerchi, punti e anche colorare intere zone dello schermo.
Questo capitolo vi introdurrà nel mondo della programmazione grafica del PC. Alcuni programmi e subroutine che qui compaiono possono essere incorporati nei vostri programmi, mentre la maggior parte dei programmi presentati come esempi sono abbastanza brevi e facilmente ricopiabili nel PC. Potrebbe essere interessante provare questi programmi nel corso della lettura.

## 9.1 I modi grafici

Il PC dispone di due modi grafici: a media risoluzione e ad alta risoluzione. In entrambi i modi, l'intero schermo è suddiviso in 200 righe, mentre, per quanto riguarda le colonne, sono 320 in media risoluzione e 640 in alta risoluzione.

### COORDINATE E PIXEL

Le righe e le colonne suddividono lo schermo in una griglia di coordinate: le istruzioni di un programma grafico si riferiscono a queste coordinate per localizzare uno specifico punto o una vasta area dello schermo. Il punto che corrisponde ad una particolare coordinata è chiamato *pixel* (il nome è la contrazione di *picture cell*). Le Figure 9.1 e 9.2 mostano il si-

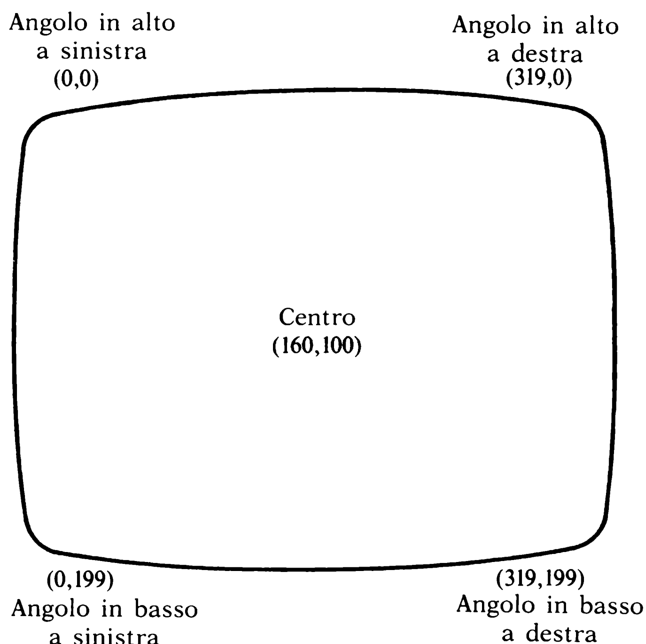



**Figura 9.1** Il sistema di coordinate nel modo Grafico a media risoluzione

stema di coordinate per i due modi, a media ed alta risoluzione.
L'angolo in alto a sinistra corrisponde alla coppia di coordinate (0,0); in tutte le istruzioni BASIC per la grafica le coordinate dello schermo devono comparire tra parentesi tonde e separate da una virgola, prima il valore relativo alla colonna e poi quello relativo alla riga. Osservate che questo sistema è l'esatto opposto di quello che viene usato nel modo Text, in cui ci si riferisce prima alla riga e poi alla colonna: è, questa, una differenza decisa intenzionalmente, poiché le coordinate sullo schermo grafico sono derivate dalle coordinate cartesiane usate in geometria, mentre le coordinate per la visualizzazione di testi usano il sistema di coordinate presente sulla maggior parte dei terminali video, compreso il monitor moncromatico IBM.
Per tenere separata la manipolazione di testi e di grafica anche sullo stesso schermo (o sulla stessa pagina di schermo), il PC usa due differenti "puntatori" alle posizioni dello schermo: in modo Grafico, l'ultimo punto a cui si è fatto riferimento, indicato con LRP (*Last Referenced Point*) è quello individuato dalle coordinate utilizzate nell'ultima istruzione grafica BASIC e differisce nettamente dalla posizione del cursore in modo Text, che invece indica il punto in cui è stato visualizzato l'ultimo carattere. I due puntatori non interferiscono l'uno con l'altro, anche se alcune istruzioni possono indirizzarli contemporaneamente.

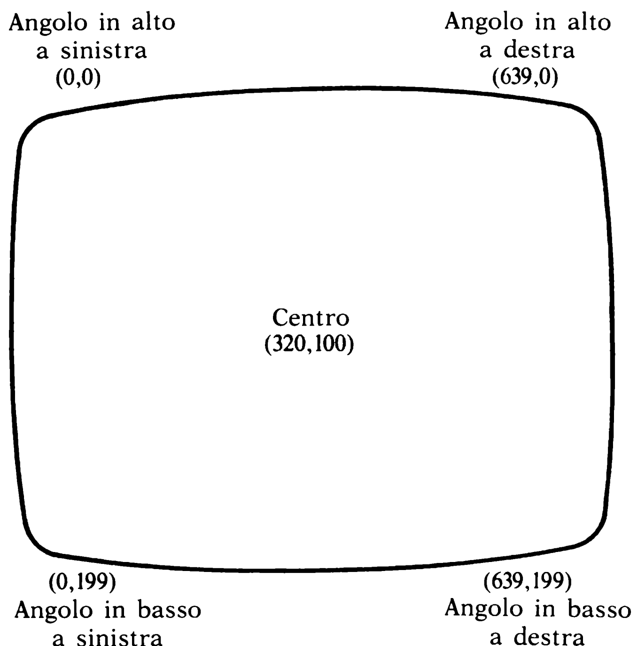



**Figura 9.2** Il sistema di coordinate nel modo Grafico ad alta risoluzione

## L'ISTRUZIONE SCREEN

Come ricorderete dal Capitolo 6, l'istruzione SCREEN (da non confondere con la funzione SCREEN, che rileva gli attributi delle locazioni dello schermo) cancella lo schermo e predispone i modi dello schermo. Screen 0, il modo Text a colori, permette di programmare più di uno schermo per volta; il modo Grafico, invece, impegna tutta la memoria disponibile nella scheda Colore/Grafica e perciò può lavorare su un solo schermo per volta.
L'istruzione:

```
SCREEN 1
```

cancella lo schermo e crea una pagina a colori a media risoluzione. Il valore di LRP è (160,100), cioè il centro dello schermo. Se volete la media risoluzione, ma non i colori, dovete inviare l'istruzione:

```
SCREEN 1,1
```

Ogni valore diverso da zero che segue la virgola elimina il colore: questa

è un'opzione utile se il vostro monitor è in bianco e nero, perché dona alle immagini maggiore definizione.
Per lo schermo ad alta risoluzione usate l'istruzione:

```
SCREEN 2
```

che automaticamente disabilita il colore sullo schermo e assegna a LRP il valore (320,100).

## 9.2 Come predisporre lo schermo a colori

Il modo Grafico a media risoluzione consente la scelta di un colore tra i 16 per il fondo, colore che copre lo schermo come uno scenario. Nella Tabella 9.1 è riportato l'elenco dei possibili colori per il fondo.

**Tabella 9.1** Colori per il fondo

| Numero | Colore | Numero | Colore |
|---|---|---|---|
| 0 | Nero | 8 | Grigio |
| 1 | Blu | 9 | Azzurro |
| 2 | Verde | 10 | Verde chiaro |
| 3 | Cyan | 11 | Cyan chiaro |
| 4 | Rosso | 12 | Rosso chiaro |
| 5 | Magenta | 13 | Magenta chiaro |
| 6 | Marrone | 14 | Giallo |
| 7 | Bianco | 15 | Bianco brillante |

I sei possibili colori per il testo vengono usati, nella grafica a media risoluzione, per disegnare punti, linee e figure: questi sei colori sono suddivisi in due tavolozze separate, ciascuna delle quali contiene anche un colore di fondo, come potete vedere nella Tabella 9.2. Può essere utilizzata

**Tabella 9.2** Tavolozze dei colori per la grafica a media risoluzione

| Tavolozza 0 (default) | | Tavolozza 1 | |
|---|---|---|---|
| **Numero** | **Colore** | **Numero** | **Colore** |
| 0 | Fondo | 0 | Fondo |
| 1 | Verde | 1 | Cyan |
| 2 | Rosso | 2 | Magenta |
| 3 | Marrone | 3 | Bianco |

solo una tavolozza per volta e ciò riduce a quattro il numero dei colori per lo schermo a media risoluzione.

## COME USARE L'ISTRUZIONE COLOR

Nel modo Grafico a media risoluzione, l'istruzione COLOR è dotata di due parametri: il colore del fondo (da 0 a 15) ed il selettore della tavolozza (0 o 1). Questi valori, in realtà, possono essere maggiori dei limiti fissati: potete scegliere, cioè, il colore 29 per il fondo e la tavolozza numero 7, ma il BASIC considera solo i quattro bit di ordine più basso per il valore del colore (come se utilizzasse 29 MOD 4, cioè 13) e gli ultimi due per la tavolozza, prendendo la tavolozza 0 per tutti i numeri pari e la 1 per tutti i numeri dispari.
L'istruzione:

```
COLOR 5
```

perciò colora il fondo di magenta; il BASIC attiva per default la tavolozza 0 se non viene specificato altrimenti nell'istruzione COLOR.
Il testo appare in colore marrone se è in uso la tavolozza 0, mentre in colore bianco se è in uso la tavolozza 1.
Nel modo Grafico ad alta risoluzione il BASIC non esegue assolutamente l'istruzione COLOR, ma visualizza un messaggio d'errore; perciò se intendete scrivere programmi in cui vengano utilizzati entrambi i modi, ponete molta attenzione a che le eventuali istruzioni COLOR compaiano solo in subroutine o funzioni per non disturbare l'esecuzione del programma.

## COME CAMBIARE LE TAVOLOZZE DEI COLORI

Per cambiare le tavolozze dei colori senza cambiare il colore del fondo, tralasciate il parametro relativo a questo nell'istruzione COLOR, sostituendolo con una virgola ed indicate il nuovo valore della tavolozza, come segue:

```
COLOR ,0
```

Una virgola che segue immediatamente l'istruzione COLOR fa sì che il BASIC lasci invariato il colore del fondo e cambi solo la tavolozza dei colori. Ogni volta che l'istruzione COLOR cambia la tavolozza, tutto il disegno (grafica e testo) passa dal vecchio colore a quello corrispondente nella nuova tavolozza: tutte le parti di schermo color magenta (tavolozza 1) diventano rosse con la tavolozza 0, il bianco diventa marrone e così via.

## 9.3 Come disegnare i punti: le istruzioni PSET e PRESET

Potete disegnare punti sullo schermo definendo il colore di un singolo pixel per mezzo delle istruzioni PSET e PRESET: queste due istruzioni hanno un significato opposto e funzionano come una matita e una gomma. PSET accende il pixel con il colore scelto, mentre PRESET riporta il pixel al colore del fondo, "cancellandolo" così dallo schermo.
Ecco un breve programma, che usa uno schermo ad alta risoluzione, SCREEN 2, in cui l'istruzione PSET nella linea 30 disegna punti casualmente sullo schermo, mentre l'istruzione PRESET cancella, pure casualmente, alcuni punti. Si noti che dopo l'esecuzione del programma dovete inviare il comando SCREEN 1 per ritornare al modo Grafico a media risoluzione.

```
10 SCREEN 2
20 CLS
30 PSET (RND(1)*639,RND(1)*199)
40 PRESET (RND(1)*639,RND(1)*199)
50 GOTO 30
```

### COME DISEGNARE CON VARI COLORI

Come abbiamo detto PSET e PRESET, nella loro forma più semplice sono opposte, perché utilizzano due colori opposti di default per disegnare: PSET colora il pixel sempre nel colore 3 (marrone o bianco a seconda della tavolozza) nel modo Grafico a media risoluzione, mentre PRESET usa per default il colore 0, cioè il colore del fondo predefinito con l'istruzione COLOR. Quando però utilizzate le due istruzioni con colori specifici, cioè indicati esplicitamente, esse si comportano identicamente; se PSET e PRESET usano lo stesso colore, le due istruzioni:

```
PSET (120,90),3
```

e

```
PRESET (120,90),3
```

sono funzionalmente equivalenti.
Il valore che indica il colore può variare tra 0 e 3 e si riferisce ai colori elencati nella Tabella 9.2. Segue ora un programma che usa PSET e PRESET per disegnare punti casuali con colori casuali:

```
10 SCREEN 1
20 CLS
30 PSET (RND(1)*319,RND(1)*199),RND(1)*3
40 PRESET (RND(1)*319,RND(1)*199),RND(1)*3
50 GOTO 30
```

## 9.4 Controllo dei punti dello schermo: la funzione POINT

In alcune occasioni potrete dover controllare il colore di un particolare pixel. La funzione POINT restituisce il colore di ogni punto dello schermo: in modo Grafico a media risoluzione questo è un valore compreso tra 0 e 3, corrispondente ai colori della Tabella 9.2, mentre ad alta risoluzione il valore è 0 per il nero e 1 per il bianco.
Il seguente programma usa le istruzioni PSET e POINT per disegnare punti casuali con colori casuali, finché PSET non cerca di scrivere sopra ad un pixel già colorato; a quel punto il programma scrive il numero di punti che sono stati disegnati:

```
10 KEY OFF
20 SCREEN 1:CLS
30 CNT=1
40 X=RND(1)*319:Y=RND(1)*199
50 IF (POINT(X,Y)<>0) THEN PRINT CNT:END
60 PSET (X,Y), RND(1)*3
70 CNT=CNT+1
80 GOTO 40
```

Questa funzione è particolarmente utile nell'animazione grafica per rilevare le collisioni tra due oggetti: dopo aver calcolato la posizione che un oggetto in movimento sta per occupare, usate POINT per controllare se in quella posizione il colore non è quello del fondo. La presenza di un qualsiasi colore diverso dal fondo indica che un altro oggetto sta già occupando quella posizione e perciò si verifica la collisione. La funzione POINT ha una variante, descritta più avanti, che serve per determinare l'ultimo punto di riferimento (LRP).

## 9.5 Coordinate assolute e relative

Ogniqualvolta le coordinate appaiono in un'istruzione grafica, esse possono essere assolute o relative: le coordinate assolute si riferiscono ad una posizione dello schermo la cui colonna e riga sono fissate, cioè, a partire dalla riga e dalla colonna 0; le coordinate assolute (120,19) si riferiscono a quel punto che si trova nella centoventunesima colonna e nella ventesima riga dello schermo.
Le coordinate relative, d'altra parte, esprimono la distanza, in termini di righe e colonne, dall'ultimo punto di riferimento (LRP). Se un'istruzione PSET ha appena colorato il pixel nella posizione (100,100) — in coordinate assolute — potete colorare il pixel che si trova nella posizione assoluta, (90,120), in questo modo:

```
PSET (90,120)
```

oppure, usando le coordinate relative:

```
PSET STEP (-10,20)
```

L'istruzione STEP compie un'operazione "grafica" in funzione dell'LRP, in questo caso muove dieci pixel a sinistra e venti pixel in basso rispetto all'ultimo punto di riferimento prima di disegnare. Questa caratteristica rende molto flessibile la grafica del PC, poiché è sufficiente modificare una coordinata assoluta per definire anche una forma molto complessa sullo schermo. Il prossimo paragrafo spiegherà in dettaglio l'uso delle coordinate relative.

## 9.6 Linee e rettangoli: l'istruzione LINE

L'istruzione LINE, come suggerisce il nome, disegna linee rette che congiungono due coppie di coordinate: potete indicare il punto di partenza, o potete utilizzare l'LRP come punto di partenza implicito. In ogni caso, l'istruzione richiede che voi specifichiate in quali coordinate disegnare. Nelle seguenti istruzioni, l'LRP è il punto di partenza implicito per ognuna delle tre linee:

```
10 SCREEN 1
20 CLS
30 LINE -(319,0)
40 LINE -(160,199)
50 LINE -(160,100)
```

Questo breve programma disegna un triangolo nel colore 3, cioè bianco o marrone a seconda della tavolozza usata: l'istruzione LINE, infatti usa per default il colore 3; i trattini che precedono le coordinate indicano che il punto di inizio è LRP.
Se volete indicare sia il punto di partenza che quello di arrivo, aggiungete le coordinate del punto di inizio tra la parola LINE ed il trattino. Ecco un breve esempio dell'uso di LINE con uno specifico punto di partenza:

```
10 SCREEN 2          'Predispone l'alta risoluzione
20 CLS
30 FOR ROW=0 TO 199
40 LINE (0,ROW)-(ROW,199)
50 NEXT ROW
```

## L'ISTRUZIONE LINE CON COORDINATE RELATIVE

Le coordinate relative possono sembrare difficili da comprendere ma in realtà il concetto di fondo è molto semplice. Se un programma deve visualizzare delle figure in punti diversi dello schermo, le coordinate relative sono più comode da usare di quelle assolute: il motivo sta nel fatto che il calcolo delle coordinate assolute in BASIC richiede molto più tempo di quello delle coordinate relative, anche se vengono programmate come variabili. Se usate, perciò, coordinate relative, le istruzioni di grafica risulteranno molto più facili da leggere e più veloci da eseguire.
Il seguente programma disegna una stella a cinque punte, usando le coordinate relative ed il punto di partenza implicito; gli spazi bianchi aggiuntivi facilitano la lettura delle linee di istruzioni, ma non sono essenziali per il funzionamento del programma.

```
10 SCREEN 1
20 CLS
30 COLOR 12,1 'Fondo rosso brillante, tavolozza 1
70 GOSUB 1000 'Disegna la stella
80 GOTO 80
1000 REM Subroutine per il disegno della stella
1010 LINE - STEP (3,3)
1020 LINE - STEP (6,0)
1030 LINE - STEP (-6,3)
1040 LINE - STEP (3,3)
1050 LINE - STEP (-6,-3)
1060 LINE - STEP (-6,3)
1070 LINE - STEP (3,-3)
1080 LINE - STEP (-6,-3)
1090 LINE - STEP (6,0)
1100 LINE - STEP (3,-3)
1110 RETURN
```

Lo stesso programma potrebbe essere scritto con coordinate assolute e con uno specifico punto di inizio, ma le istruzioni LINE dovrebbero contenere delle variabili che, a loro volta, dovrebbero essere ricalcolate ogni volta che si volesse disegnare l'oggetto in un'altra posizione dello schermo o ripeterlo un certo numero di volte.
In breve, vi conviene sempre utilizzare questo metodo con coordinate relative a un punto di partenza implicito se volete creare una figura, che sia sempre la stessa, formata da linee consecutive o comunque collegate; se qualche linea non fosse collegata alle altre, utilizzate un punto di partenza esplicito nell'istruzione LINE.
Per dimostrare la flessibilità delle coordinate relative e del punto di partenza implicito nell'istruzione LINE, modificate il programma precedente con l'aggiunta delle seguenti linee:

```
40 COL=INT(RND*319)
50 ROW=INT(RND*199)
60 PSET (COL,ROW),0
80 GOTO 40
```

Ora provate a far eseguire il programma modificato: l'intero schermo si riempirà di stelle. Le variabili COL e ROW variano casualmente (usando la funzione RND) e l'istruzione PSET di linea 60 le utilizza disegnando un punto con il colore del fondo, assegnando il nuovo valore a LRP. La linea 70, che era già stata scritta in precedenza, esegue la subroutine che crea la stella; quando la prima stella è terminata, il processo viene ripetuto indefinitamente saltando indietro fino alla linea 40.

## COME SCEGLIERE I COLORI NELL'ISTRUZIONE LINE

L'istruzione LINE ha una sintassi molto simile a PSET e PRESET per la scelta dei colori con cui disegnare: il numero del colore (da 0 a 3) compare dopo l'ultima (o la sola) coppia di coordinate dell'istruzione, separato da queste per mezzo di una virgola. Perciò, l'istruzione:

```
LINE (0,0)-(180,180),1
```

disegnerà una linea tra i due punti assoluti (0,0) e (180,180) con il colore 1. Se non viene indicato il numero del colore, il BASIC usa il colore di default 3 piuttosto che l'ultimo colore scelto con un'istruzione LINE, PSET o PRESET; siate perciò espliciti nell'indicare il colore, a meno che non vi importi di usare il colore di default.
Nel modo Grafico ad alta risoluzione il colore di default è 1, cioè bianco. Usando lo 0 o un'espressione uguale a zero per indicare il colore, potrete

visualizzare uno o più punti in nero, permettendovi così di cancellare selettivamente alcune parti di schermo precedentemente disegnate.

## COME DISEGNARE RETTANGOLI CON L'ISTRUZIONE LINE

Un'altra opzione dell'istruzione LINE, spesso molto utile, è l'opzione "rettangolo" (*box*): con questa potete creare delle barre, sottolineare parti di testo, disegnare un bordo intorno allo schermo ed inventare nuove configurazioni basate sull'uso dei rettangoli.
L'opzione rettangolo è molto semplice da aggiungere all'istruzione LINE:

```
LINE (20,20)-(100,100),,B
```

In questo esempio le coordinate (20,20) e (100,100) identificano gli angoli opposti di un rettangolo, invece che i due estremi di una linea. In questo caso non è stato specificato il colore, come si può notare dalla presenza delle due virgole prima della B: se presente, infatti, il colore avrebbe dovuto trovarsi esattamente tra quelle virgole.
La lettera B specifica l'opzione rettangolo: potete ancora scegliere tra l'uso di coordinate assolute o relative, e nell'ultimo caso l'LRP sarà il punto di partenza implicito per il rettangolo. L'opzione può essere migliorata con la possibilità di riempire i rettangoli con un colore a scelta.

## OPZIONE DI RIEMPIMENTO DEI RETTANGOLI

La combinazione delle lettere B e F al termine dell'istruzione LINE non solo disegna il rettangolo, ma lo riempie anche con il colore stabilito:

```
10 SCREEN 1
20 CLS
30 COLOR 0,1 'Fondo nero, tavolozza 1
40 LINE (140,0)-(180,199),2,BF
50 GOTO 50
```

Questo programma crea un rettangolo che si estende dall'alto verso il basso dello schermo. Questa opzione è di esecuzione decisamente veloce, soprattutto in relazione al numero di dati che devono essere posti nella memoria grafica: il riempimento di tutto lo schermo richiede all'incirca un decimo di secondo; naturalmente la stessa operazione su di un'area più piccola richiede un tempo più breve.

### COME CAMBIARE LO STILE NELL'ISTRUZIONE LINE

Con l'opzione "stile" dell'istruzione LINE si ottengono linee tratteggiate o a punti. Il valore da fornire è una maschera di 16 bit: ogni bit della maschera corrisponde ad un pixel della linea. Quando un punto dev'essere disegnato, viene esaminato il bit corrente nella maschera: se questo vale 1, il pixel viene illuminato con il colore scelto, altrimenti non viene modificato.
L'istruzione LINE usa i successivi bit della maschera per determinare quali pixel debbono essere colorati: quando raggiunge il sedicesimo bit, ritorna all'inizio della maschera per il pixel che segue. I bit della maschera continuano ad essere confrontati con i pixel della linea da disegnare finché la linea non è completa. Potete anche disegnare un rettangolo con differenti motivi sui quattro lati definendo quattro diversi stili, uno per ogni lato:

```
Lato 1 - 0 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1 = &HAAAA
Lato 2 - 0 0 1 1 0 0 1 1 0 0 1 1 0 0 1 1 = &H3333
Lato 3 - 1 1 1 1 0 0 0 0 1 1 1 1 0 0 0 0 = &HF0F0
Lato 4 - 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 = &HFF00
```

Ed ecco il programma che disegna il quadrato con l'opzione stile:

```
10 KEY OFF:SCREEN 1:CLS
20 LINE -(160,20),1,,&HAAAA
30 LINE -(240,20),2,,&H3333
40 LINE -(240,100),3,,&HF0F0
50 LINE -(160,100),2,,&HFF00
```

Si possono anche combinare le due opzioni, B (rettangolo) e stile, nell'istruzione LINE per disegnare un rettangolo con quattro lati uguali. Basta specificare sia B che lo stile scelto; non è permesso, però, il riempimento dei rettangoli (opzione BF) in combinazione con l'opzione stile: un tentativo in tale senso provocherebbe un errore di sintassi.

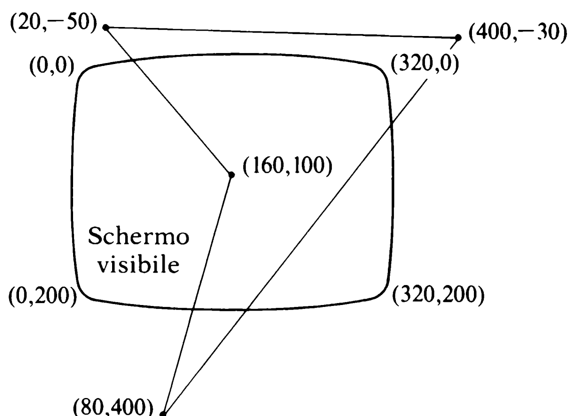
## 9.7 Taglio delle linee

Spesso lo schermo del PC risulta troppo piccolo per mostrare un'intera figura nelle dimensioni che ci servono: questo induce una parte del disegno a "cadere" al di fuori di uno dei lati dello schermo. L'istruzione LINE usa una tecnica, chiamata "taglio delle linee" per selezionare e disegnare solo quella parte delle linee o dei rettangoli che rientra nell'area

dello schermo: semplicemente, LINE interrompe ogni linea che fuoriesce dai limiti della superficie visibile dello schermo. Questa tecnica è analoga a quanto si potrebbe fare disegnando una figura su di un grande foglio di carta per poi ritagliarne con un paio di forbici le parti essenziali. L'esempio riportato qui sotto illustra il taglio delle linee: osservate che l'LRP di una linea tagliata rimane nell'estremo della linea, fuori dallo schermo. La seconda linea nell'esempio ha entrambi gli estremi all'esterno dello schermo, ma viene visualizzata correttamente la parte di linea che lo attraversa; la terza non passa mai per lo schermo e perciò non risulta visibile; la quarta istruzione LINE usa come LRP l'estremo della terza linea.

```
10 KEY OFF:SCREEN 1:CLS
20 LINE -(80,400),1
30 LINE -(400,-30),2
40 LINE -(20,-50),3
50 LINE -(160,100),3
```

Qui sotto è riportata la vista complessiva del disegno creato dal programma precedente: solo le linee che cadono all'interno dell'area dello schermo saranno visibili dopo l'esecuzione del programma.




Tutti i comandi grafici presentati nel resto del capitolo usano la tecnica di taglio delle linee: tutte le parti di oggetti che cadono al di fuori dei limiti dello schermo vengono tagliate ai bordi dello stesso.

# 9.8 L'istruzione CIRCLE

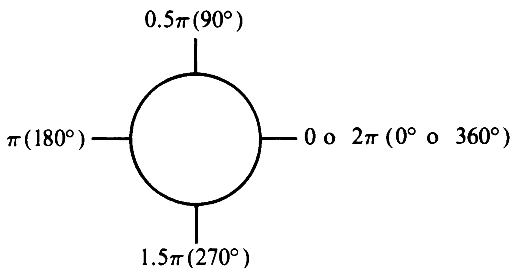
L'istruzione CIRCLE, disponibile solo in Advanced BASIC, è uno strumento grafico molto utile: con essa potete disegnare cerchi, ellissi o archi nei colori disponibili; come per tutte le altre istruzioni grafiche in BASIC, potete usare sia le coordinate assolute che quelle relative.
Per farvi un'idea di come funzioni l'istruzione CIRCLE, provate in modo diretto i comandi:

```
SCREEN 1
CIRCLE STEP (0,0),40,1
```

Vedrete apparire un cerchio di colore cyan all'incirca nel centro dello schermo. Avendo inserito la parola STEP, l'istruzione usa le coordinate relative, cioè si muove a 0 colonne e 0 righe dall'LRP, che era stato posto uguale a (160,100) dal comando SCREEN 1. Il parametro che segue le coordinate è il raggio del cerchio, in questo caso 40 pixel lungo l'asse orizzontale; dopo il raggio è il parametro del colore, separato da una virgola.
Dopo aver concluso il cerchio, l'istruzione assegna a LRP le coordinate indicate, in questo caso (160,100).
Tentate qualche esperimento con differenti coordinate iniziali o con diversi raggi e poi passate a considerare altre caratteristiche dell'istruzione CIRCLE.

### COME DISEGNARE ARCHI

Un arco è una parte di cerchio: per esprimere l'angolo sotteso da un arco il PC usa i radianti, cioè gli angoli sono dati in funzione di pi greco ($\pi$).
Nel disegno potete vedere quali valori assumano gli angoli in radianti ed i loro corrispettivi in gradi:



Un'istruzione che disegni solo metà di una circonferenza deve usare un angolo di inizio ed uno di arrivo per l'arco. Il breve programma che segue disegna un arco con centro nel punto di coordinate assolute (40,40) con raggio di 16 pixel dall'angolo 0 a 3.14159 (=$\pi$) radianti.

```
5 PI=3.14159
10 SCREEN 1:COLOR 0,0
20 CLS
30 CIRCLE (40,40),16,,0,PI
```

Gli ultimi due parametri nella linea 30, cioè 0 ed il valore della variabile PI, indicano l'angolo di partenza e quello di arrivo del semicerchio. Ricordate che le coordinate indicate nell'istruzione CIRCLE si riferiscono al centro dell'arco o del cerchio disegnato; osservate anche che in questa istruzione CIRCLE non abbiamo scelto il colore, ma abbiamo utilizzato il colore di default, cioè il marrone.

## COME DISEGNARE UN RAGGIO

Il semicerchio dell'esempio precedente potrebbe essere disegnato con l'aggiunta di due raggi, uno che delimita l'angolo iniziale (0 radianti) e l'altro l'angolo finale ($\pi$ radianti). Per far questo, dobbiamo usare gli stessi angoli di partenza e di arrivo, ma esprimerli come numeri negativi. Quando il BASIC trova un'istruzione CIRCLE con un angolo negativo, usa il valore assoluto come angolo di delimitazione dell'arco ed il segno meno per indicare che dev'essere disegnato un raggio con l'angolo indicato. Un'istruzione come:

```
30 CIRCLE (40,40),16,,0,-1*PI
```

disegna un semicerchio ed un raggio dall'estremo sinistro di questo fino al centro del cerchio in (40,40); osservate cosa accade se utilizzate anche il valore −0, come valore iniziale dell'angolo:

```
30 CIRCLE (40,40),16,,-0,-1*PI
```

Notate che il valore −0 viene ignorato se utilizzato come angolo di un arco: avreste dovuto usare, in sua vece, il valore −2*PI, che avrebbe prodotto un angolo equivalente.

## TRACCIAMENTO DEL DISEGNO

L'istruzione CIRCLE disegna sempre in senso antiorario a partire dall'angolo iniziale




Questo può non essere importante se volete disegnare un cerchio intero, ma può influire su come e dove viene disegnato un arco. Nell'esempio del semicerchio, il disegno inizia dall'angolo di 0 radianti; se venissero invertiti l'angolo iniziale e quello finale nell'istruzione, CIRCLE disegnerebbe in senso antiorario da $\pi$ radianti a 0 radianti, creando cioè la metà inferiore del cerchio invece che quella superiore.

## COME DISEGNARE ELLISSI

Potete utilizzare l'istruzione CIRCLE anche per disegnare ellissi, semplicemente cambiando il rapporto tra l'asse verticale e quello orizzontale.
I due modi grafici del PC utilizzano due diversi valori di questo rapporto per disegnare archi e cerchi: per default l'istruzione CIRCLE usa il valore 5:6 in media risoluzione ed il valore 5:12 in alta risoluzione.
Se ad esempio disegnate un cerchio con raggio 16, questo significa che l'effettiva distanza dal centro del cerchio è di 16 pixel lungo l'asse orizzontale, mentre lungo l'asse verticale i pixel saranno 13, cioè cinque sesti di 16 (arrotondando il risultato). Se invece il cerchio viene disegnato in alta risoluzione, un raggio di 16 pixel lungo l'asse orizzontale significherà $5/12*16 \simeq 7$ pixel lungo l'asse verticale.

## L'ISTRUZIONE CIRCLE CON L'INDICAZIONE DEL RAPPORTO TRA GLI ASSI

Per cambiare il rapporto tra gli assi dell'ellisse dovete aggiungere il nuovo valore al termine dell'istruzione CIRCLE. Le seguenti istruzioni disegnano un cerchio con un rapporto 1:1:

```
10 SCREEN 1:COLOR 1,0
20 CLS
30 CIRCLE STEP (0,0),25,,,,1
```

in modo che i punti della circonferenza passino a 25 pixel di distanza dal centro lungo entrambi gli assi, orizzontale e verticale.
Diminuendo il valore del rapporto si ottiene un'ellisse allungata orizzontalmente; al contrario, aumentando il rapporto 1, l'ellisse si distende lungo l'asse verticale. Dopo aver provato l'esempio precedente, cambiate la linea 30:

```
30 CIRCLE STEP (0,0),25,,,,1/5
```

In questo modo il raggio è ancora di 25 pixel lungo l'asse orizzontale, ma solo un quinto di 25, cioè 5, lungo l'asse verticale.
Per disegnare un'ellisse disposta verticalmente utilizzate un rapporto maggiore di 1, come potete vedere qui di seguito:

```
30 CIRCLE STEP (0,0),25,,,,5
```

Il valore 5 del rapporto è esattamente l'inverso della relazione precedente: ora l'ellisse ha un raggio di 25 pixel lungo l'asse verticale, ma di soli 5 pixel, cioè un quinto di 25, lungo l'asse orizzontale.

## 9.9 L'istruzione PAINT

Se avete provato ad utilizzare l'istruzione LINE con l'opzione di riempimento presentata all'inizio di questo capitolo, vi sarete probabilmente chiesti come fare per riempire aree di schermo circolari, poligonali, o comunque non rettangolari: la risposta a ciò è l'istruzione PAINT, disponibile solo in Advanced BASIC, che riempie una qualsiasi area chiusa dello schermo indipendentemente dalla forma.
Il seguente programma disegna una stella a cinque punte usando il colore numero 3 della tavolozza 1 (cioè, il bianco); dopo che è stato disegnato il contorno della stella, l'istruzione PAINT ne riempie l'interno con il colore 2 (magenta).

```
10 SCREEN 1
20 CLS
30 COLOR 12,1 'Fondo rosso, tavolozza 1
40 COL=INT(RND*319)
50 ROW=INT(RND*199)
60 PSET (COL,ROW),0 'Definisce il primo LRP
70 GOSUB 1000 'Disegna la stella
80 PAINT (COL,ROW+1),2,3 'La colora (PAINT) di
magenta
90 GOTO 40 'Disegna a partire da altre coordinate
```

```
1000 REM Subroutine di disegno
1010 LINE - STEP (3,3)
1020 LINE - STEP (6,0)
1030 LINE - STEP (-6,3)
1040 LINE - STEP (3,3)
1050 LINE - STEP (-6,-3)
1060 LINE - STEP (-6,3)
1070 LINE - STEP (3,-3)
1080 LINE - STEP (-6,-3)
1090 LINE - STEP (6,0)
1100 LINE - STEP (3,-3)
1110 RETURN
```

L'istruzione PAINT alla linea 80 si riferisce a coordinate che stanno all'interno del contorno della stella: osservate che l'istruzione PAINT non può utilizzare coordinate relative e perciò ricordatevi sempre di inserire coordinate assolute per evitare di incorrere nell'errore ILLEGAL FUNCTION CALL.
Appena dopo le coordinate, inserite il colore scelto, in questo caso il 2 (magenta); il terzo parametro, 3, indica il colore del contorno: l'istruzione PAINT colora la figura fino a quando incontra il contorno e lo riconosce attraverso il colore.
Sia la tinta che il colore del contorno hanno come valore di default 3, se non viene indicato altrimenti: nell'esempio, perciò, il valore del contorno è stato inserito solo al fine di meglio illustrare l'uso dei parametri.

## COME DEFINIRE L'AREA DA COLORARE

Perché l'istruzione PAINT agisca correttamente dovete sempre assicurarvi che il valore di LRP si riferisca ad un punto interno all'area da riempire; se LRP si trova sul contorno della figura o interamente al di fuori, non verrà riempita la figura, ma tutto il resto dello schermo, tranne l'area che intendevate colorare.
Inoltre è necessario che il contorno dell'area da colorare sia chiuso, perché l'istruzione PAINT cambia il colore di tutti i pixel che incontra, fino al contorno; se una parte di questo manca, l'istruzione agisce anche sul resto dello schermo, portando a risultati imprevedibili specialmente quando, nella parte rimanente dello schermo, si incontrano pixel dello stesso colore del contorno.
Ecco ora un programma che disegna un paesaggio usando l'istruzione PAINT, CIRCLE ed altre, illustrate in questo capitolo:

```
10 SCREEN 1:COLOR 0,1 'Ris. media, tavolozza 1
20 CLS
```

```
30 PSET (0,100) 'Definisce il punto iniziale
40 FOR I=20 TO 320 STEP 20
50 LINE -(I,100+RND*10),2 'Disegna le linee del ter
reno
60 NEXT I
70 PAINT (0,199),1,2 'Le colora (PAINT) con il colo
re 1
80 FOR I=1 TO 40
90 PSET (RND*319,RND*108) 'Disegna 40 stelle
100 NEXT I
110 CIRCLE (200,20),18,2  'Disegna la luna
120 PAINT (200,20),2,2 'La riempie con il colore 2
130 GOTO 130
```

Le linee da 40 a 60 creano il paesaggio, che si estende lungo lo schermo (a media risoluzione) di 20 punti alla volta; la funzione RND crea un orizzonte accidentato variando leggermente la coordinata verticale nell'istruzione LINE della linea 50.
La linea 70 serve a colorare il paesaggio, dal fondo dello schermo fino alla linea dell'orizzonte; le linee dalla 80 alla 100 disegnano, distribuendole in modo casuale, 40 stelle al di sopra dell'orizzonte per rendere più suggestivo l'effetto.
Per completare il disegno, l'istruzione CIRCLE di linea 110 crea una luna sopra l'orizzonte, che viene colorata con il colore numero 2, dall'istruzione PAINT in linea 120.

## MOTIVI DI RIEMPIMENTO

Un oggetto può essere riempito con uno schema o un disegno continuamente ripetuto, piuttosto che con un singolo colore; un'area viene riempita con numerose "mattonelle", che spesso creano risultati finali molto interessanti.
Questo tipo di riempimento viene specificato nell'istruzione PAINT quando il terzo parametro, il colore con cui dipingere, viene sostituito da un'espressione di tipo stringa. La stringa, che definisce la maschera su cui verrà costruita la mattonella, deve avere la seguente forma:

CHR$(&H*nn*) + CHR$(&H*nn*) + CHR$(&H*nn*) + ...

La maschera può essere lunga da 1 a 64 caratteri, ognuno dei quali definisce un valore di un byte che rappresenta una riga del motivo: questa maschera viene poi usata ripetutamente per riempire (sempre con l'istruzione PAINT) l'area richiesta.
Nel modo grafico ad alta risoluzione ogni bit della maschera corrisponde

ad un pixel dello schermo: viene perciò disegnato un punto per ogni posizione della maschera in cui il valore sia 1.
Nella grafica a media risoluzione, invece, occorrono due bit per definire il colore di ogni pixel dello schermo; la scelta dei colori dipende dalla tavolozza corrente, come indicato nella Tabella 9.2. Dal momento che due bit della maschera corrispondono ad un unico pixel dello schermo, ogni byte disegna solo quattro punti: una maschera di 64 caratteri, così, specifica una mattonella rettangolare larga 4 pixel ed alta 64.
Il paesaggio dell'esempio precedente può essere modificato per dimostrare questa nuova caratteristica dell'istruzione PAINT; definiamo la maschera per l'area del terreno come un rettangolo di 4×8 pixel. La maschera di 8 byte è riportata qui di seguito, con ogni byte suddiviso nei colori (cioè due bit per volta):

```
Motivo 0 - 10 01 01 01     CHR$(&H95)
Motivo 1 - 01 10 01 01     CHR$(&H65)
Motivo 2 - 01 01 10 01     CHR$(&H59)
Motivo 3 - 01 01 01 10     CHR$(&H56)
Motivo 4 - 01 01 01 10     CHR$(&H56)
Motivo 5 - 01 01 10 01     CHR$(&H59)
Motivo 6 - 01 10 01 01     CHR$(&H65)
Motivo 7 - 10 01 01 01     CHR$(&H95)
```

Disegneremo poi la luna con questa maschera di quattro byte:

```
Motivo 0 - 11 00 00 11     CHR$(&HC3)
Motivo 1 - 00 11 11 00     CHR$(&H3C)
Motivo 2 - 00 11 11 00     CHR$(&H3C)
Motivo 3 - 11 00 00 11     CHR$(&HC3)
```

Il programma finale diventa:

```
10  SCREEN 1:COLOR 0,1
20  CLS
30  PSET (0,100)
40  FOR I=20 TO 320 STEP 20
50  LINE -(I,100+RND*10),2
60  NEXT I
65  MOTIVO1A$=CHR$(&H95)+CHR$(&H65)+CHR$(&H59)+CHR$(&
H56)
67  MOTIVO1B$=CHR$(&H56)+CHR$(&H59)+CHR$(&H65)+CHR$(&
H95)
70  PAINT (0,199),MOTIVO1A$+MOTIVO1B$,2
80  FOR I=1 TO 40
90  PSET (RND*319,RND*108)
100  NEXT I
```

```
110  CIRCLE (200,20),18,2
115  MOTIVO2$=CHR$(&HC3)+CHR$(&H3C)+CHR$(&H3C)+CHR$(&
HC3)
120  PAINT (200,20),MOTIVO2$,2
130  GOTO 130
```

Osservate che possono insorgere problemi nel caso in cui la maschera contenga due byte consecutivi identici al colore corrente dell'oggetto da riempire: quando PAINT entra in esecuzione, stabilisce che il punto è tutto circondato dallo stesso colore e, erroneamente, termina di riempire l'oggetto.
Per sincerarvi di questo fatto, provate ad inserire nel programma precedente questa istruzione:

```
117 PAINT (200,20),CHR$(&H3C),2
```

che colora la luna con un motivo di un solo carattere, dopo di che la linea 120 cerca di ripetere l'operazione con la maschera MOTIVO2$ e si ferma prima che il lavoro sia compiuto.
Per superare questo problema, l'istruzione PAINT possiede anche un attributo di "fondo", una stringa di un byte della forma CHR$(&H*nn*).
Quando poi usate l'istruzione, la maschera può contenere due caratteri consecutivi uguali che corrispondano all'attributo del fondo, se indicato: questo permette che l'oggetto venga correttamente riempito.
Aggiungendo perciò la stringa del fondo CHR$(&H3C) alla linea 120 del programma che disegna il paesaggio, otteniamo un esatto riempimento del fondo. Ecco l'ultima versione del programma:

```
10  SCREEN 1:COLOR 0,1
20  CLS
30  PSET (0,100)
40  FOR I=20 TO 320 STEP 20
50  LINE -(I,100+RND*10),2
60  NEXT I
65  MOTIVO1A$=CHR$(&H95)+CHR$(&H65)+CHR$(&H59)+CHR$(&
H56)
67  MOTIVO1B$=CHR$(&H56)+CHR$(&H59)+CHR$(&H65)+CHR$(&
H95)
70  PAINT (0,199),MOTIVO1A$+MOTIVO1B$,2
80  FOR I=1 TO 40
90  PSET (RND*319,RND*108)
100  NEXT I
110  CIRCLE (200,20),18,2
115  MOTIVO2$=CHR$(&HC3)+CHR$(&H3C)+CHR$(&H3C)+CHR$(&
HC3)
117  PAINT (200,20),CHR$(&H3C),2
120  PAINT (200,20),MOTIVO2$,2,CHR$(&H3C)
130  GOTO 130
```

## 9.10 Trasformazioni dello schermo

In generale, le funzioni ed i comandi per la grafica finora esaminati rendono piuttosto semplice creare dei programmi che visualizzano informazioni ed animano oggetti; ci sono, però, alcune difficoltà, come ad esempio la determinazione delle coordinate, che non vanno assolutamente sottovalutate.

### DETERMINARE LE COORDINATE CON L'ISTRUZIONE WINDOW

La prima difficoltà è il sistema di coordinate fisso dello schermo: il modo Grafico a media risoluzione visualizza solo i pixel con coordinate da 0 a 319 in senso orizzontale e da 0 a 199 in senso verticale; analogamente, il modo Grafico ad alta risoluzione usa solo l'intervallo da 0 a 639 in senso orizzontale e da 0 a 199 in senso verticale. Poche delle applicazioni reali contengono valori che naturalmente cadano all'interno di questi intervalli: per far sì che i valori reali siano riportati in questo sistema fisso di coordinate, il programmatore spesso si trova costretto ad usare complesse funzioni matematiche, che impiegano grandi quantità di tempo, per trasformare i valori del mondo reale nelle appropriate coordinate dello schermo. In effetti, il solo tentativo di visualizzare lo stesso disegno nei due modi grafici, alta e media risoluzione, comporta un riadattamento delle coordinate che tenga conto delle differenza tra il numero di pixel in senso orizzontale nei due modi.
Come semplice esempio di questo problema, considerate di dover convertire un programma applicativo da un computer di altro tipo al PC-IBM: supponiamo che i due computer usino le stesse istruzioni BASIC, ma che il non-IBM abbia uno schermo di 720 pixel orizzontalmente e 320 verticalmente. Un banale tentativo di far eseguire il programma dal PC senza modifiche avrà come risultato che parte dell'output grafico che deve essere visualizzato cadrà al di fuori dei confini dello schermo del PC. Convertire il vecchio programma in uno adatto al PC richiede che voi modifichiate tutte le istruzioni grafiche controllando che tutti gli indirizzamenti a locazioni dello schermo (sia in coordinate relative che assolute) siano modificati opportunamente in relazione alle dimensioni dello schermo del PC-IBM: è un compito lungo e soggetto ad errori.
L'istruzione WINDOW, però, vi consente di risparmiare questa fatica definendo un sistema di coordinate esterno (cioè del mondo esterno): questo si ottiene specificando una regione di forma rettangolare con le due coppie di coordinate (cioè con due punti) (x1,y1) e (x2,y2) che corrispondono agli angoli opposti del rettangolo. Abbiamo in questo modo definito una "finestra", che verrà usata per ogni trasformazione o riferimento di tutte

le operazioni grafiche dalle coordinate esterne al sistema fisso dello schermo.
Nel caso della conversione del programma accennata precedentemente, possiamo aggiungere alla versione destinata all'IBM il comando:

```
WINDOW SCREEN (0,0)-(719,319)
```

Il BASIC trasformerà così automaticamente tutti gli indirizzi grafici dal sistema di coordinate esterne (definito dalle necessità del programma applicativo) al sistema fisico di coordinate dello schermo del PC; questo lascia al BASIC la responsabilità della trasformazione delle coordinate del problema reale in funzione dei limiti dello schermo.
L'istruzione WINDOW funziona con tutte le istruzioni del tipo LINE, PSET, PRESET e CIRCLE; purtroppo, però, non agisce sulle figure create con il comando DRAW, che funziona solo nel sistema di coordinate fisico dello schermo.
L'esempio seguente disegna un cerchio usando le coordinate normali ed in seguito lo trasforma in altre coordinate esterne e lo ridisegna. Notate come il primo cerchio rimanga inalterato nonostante il cambiamento di coordinate: l'istruzione WINDOW influisce solo sulle istruzioni che la seguono e non su quelle già eseguite.

```
10 KEY OFF:SCREEN 1:CLS
20 WINDOW ' Coordinate esterne = schermo
30 CIRCLE (79,49),50
40 WINDOW SCREEN (0,0)-(719,319)
45 'Disegna lo stesso cerchio usando la nuova finestra
50 CIRCLE (79,49),50,2
```

L'attributo SCREEN all'istruzione WINDOW è facoltativo: se inserito, il BASIC usa la disposizione standard degli assi. Così il comando:

```
WINDOW SCREEN (0,0)-(100,100)
```

definisce le coordinate come riportato nella Figura 9.3.
Tralasciando l'attributo SCREEN utilizziamo il sistema di coordinate cartesiane; il comando:

```
WINDOW (0,0)-(100,100)
```

definisce il sistema di coordinate mostrato nella Figura 9.4.
L'uso delle finestre permette anche di realizzare ingrandimenti e riduzioni degli oggetti visualizzati; immaginate lo schermo del computer come se fosse l'obiettivo di una telecamera che inquadra un oggetto: se fate

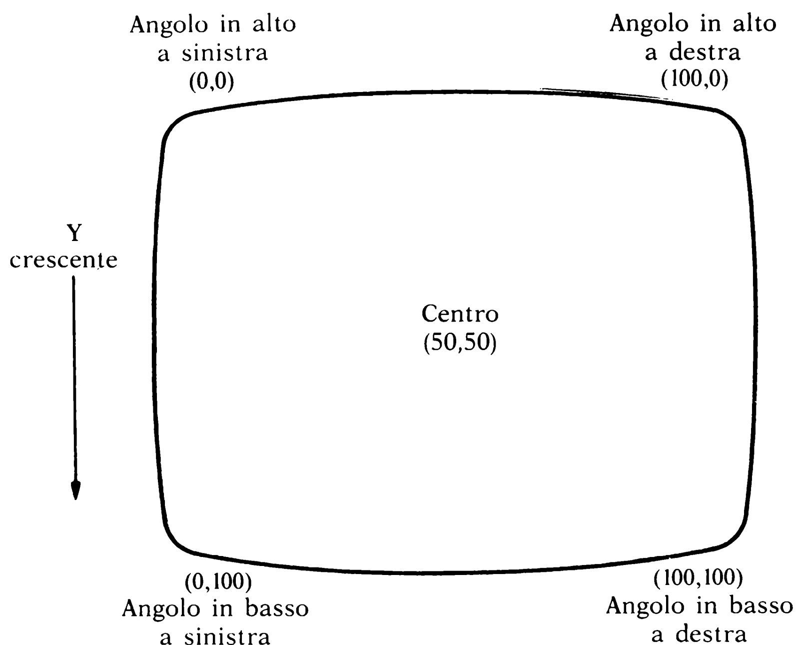



**Figura 9.3** Risultato dell'istruzione WINDOW SCREEN (0,0)–(100,100)

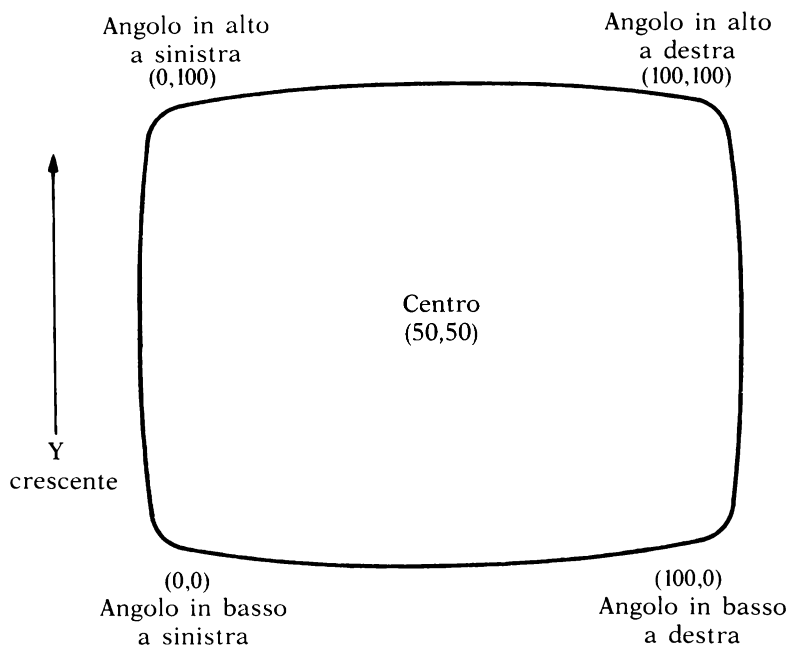



**Figura 9.4** Risultato dell'istruzione WINDOW (0,0)–(100,100)

uno zoom sull'oggetto, perdete di vista alcuni particolari periferici, ma potete esaminarne in maggiore dettaglio altri. Nello stesso modo allontanando l'immagine perdete alcuni dettagli dell'oggetto, ma avete una migliore vista d'insieme.
Ecco un programma che dimostra questi effetti di ingrandimento e riduzione ottenuti grazie all'istruzione WINDOW:

```
5 PI=3.14159
10 KEY OFF:CLS:SCREEN 1
20 CLS:PRINT "Vista normale dell'oggetto..."
30 X=160:Y=100:GOSUB 500
40 CLS:PRINT "Ingrandimento...di 5 volte.."
50 X=32:Y=20:GOSUB 500
60 CLS:PRINT "Riduzione...di 5 volte.."
70 X=800:Y=500:GOSUB 500
80 END
500 WINDOW (-X,-Y)-(X,Y)
510 CIRCLE (0,0),40,2,1.65*PI,1.35*PI,7/18
520 LINE (-20,10)-(20,-50),1,BF
530 LINE (-10,0)-(10,-40),0
540 LINE (10,0)-(-10,-40),0
550 CIRCLE (0,0),120,3,-.25*PI,-.75*PI,2
560 FOR RITARDO=1 TO 2000:NEXT RITARDO
570 WINDOW
580 RETURN
```

## UNO SCHERMO NELLO SCHERMO: L'ISTRUZIONE VIEW

Un limite delle istruzioni grafiche è l'impossibilità di riservare per esse solo una parte dello schermo. Supponiamo di voler confinare tutti gli oggetti disegnati dalle istruzioni grafiche in un'area rettangolare dello schermo, lasciando il resto libero per visualizzare del testo o anche disegni diversi che non si sovrappongano gli uni con gli altri.
Con l'istruzione VIEW possiamo definire e delimitare porzioni di schermo, dette *viewport*: sono aree rettangolari dello schermo in cui verranno visualizzate tutte le successive operazioni grafiche. Una viewport può avere qualsiasi dimensione, fino a coprire l'intero schermo, definita dalle coordinate dei due angoli opposti del rettangolo; queste coordinate, naturalmente, devono essere comprese nei limiti effettivi dello schermo (cioè, da 0 a 319 orizzontalmente e da 0 a 199 verticalmente per la media risoluzione e da 0 a 639 orizzontalmente e da 0 a 199 verticalmente per l'alta risoluzione).
Se aggiungete il valore di un colore all'istruzione VIEW, l'area definita sarà colorata completamente con quel colore; nello stesso modo, un colo-

re per l'attributo contorno disegnerà un bordo con quel colore intorno alla viewport. Si possono definire numerose viewport, ma solo l'ultima rimane attiva. Ecco un programma che dimostra l'uso di due di esse:

```
10 SCREEN 1:CLS
20 VIEW (20,20)-(99,69),1,2
30 LINE (0,0)-(319,199),0
40 LOCATE 12,4:PRINT "Viewport 1";
45 LOCATE 24,22:PRINT "Viewport 2";
50 VIEW (183,50)-(220,160),2,3
60 FOR RITARDO=1 TO 250:NEXT RITARDO
70 CLS
75 LINE -(0,0)
80 CIRCLE (0,0),20,2
100 GOTO 100
```

Osservate che la figura che dev'essere disegnata nella prima viewport è stata ridotta in modo da non uscire dall'area definita; osservate anche che l'istruzione CLS cancella solo l'area all'interno della seconda e che il cerchio viene interrotto ai bordi della viewport stessa. (Qualora voleste cancellare l'intero schermo quando una viewport è attiva, usate il comando PRINT CHR$(12)). L'istruzione VIEW permette quindi di definire uno "schermo all'interno dello schermo" e di limitare il disegno all'interno di questo nuovo schermo.
Le viewport sono definite da coordinate fisiche e non vengono influenzate da precedenti istruzioni WINDOW: in realtà, le istruzioni VIEW e WINDOW non interferiscono l'una con l'altra, ma tutte le operazioni grafiche successive utilizzeranno sia l'ultima istruzione VIEW che l'ultima WINDOW per trasformare le coordinate esterne dell'oggetto e visualizzarlo sullo schermo.
Poiché le due istruzioni non interferiscono, non ha alcuna importanza l'ordine in cui vengono eseguite. (Osservate, però, che se VIEW viene specificata in coordinate esterne invece che dello schermo, l'ordine conta moltissimo, poiché l'istruzione WINDOW modifica le coordinate esterne).
Ecco un programma dimostrativo:

```
10 'Usa VIEW e poi WINDOW
20 SCREEN 2:CLS:KEY OFF
30 VIEW (100,50)-(500,150),,1
40 WINDOW (0,0)-(100,100)
50 GOSUB 140
60 ' Ritardo
70 FOR I=1 TO 10: BEEP:NEXT I
80 ' Usa WINDOW e poi VIEW
90 SCREEN 2:CLS
100 WINDOW (0,0)-(100,100)
```

```
110 VIEW (100,50)-(500,150),,1
120 GOSUB 140
130 GOTO 130
140 ' Disegna un cerchio ed una linea
150 CIRCLE (50,50),25
160 LINE (25,25)-(75,75)
170 RETURN
```

L'istruzione VIEW ha anche l'argomento opzionale SCREEN, che specifica che le coordinate devono essere assolute e non relative; nella viewport solo i punti che rientrano nei confini della viewport saranno visibili, mentre gli altri saranno eliminati.

## COME DETERMINARE LE COORDINATE FISICHE E LE COORDINATE ESTERNE: LA FUNZIONE PMAP

Per utilizzare le capacità offerte dalle istruzioni WINDOW e VIEW, possiamo talvolta voler calcolare le coordinate fisiche dello schermo a partire dalle coordinate esterne e viceversa; la funzione PMAP consente la realizzazione della traslazione tra il sistema delle coordinate esterne come definito dall'ultima istruzione WINDOW ed il sistema delle coordinate fisiche definito dalla più recente istruzione VIEW.
La funzione PMAP restituisce una singola coordinata ad ogni esecuzione, ed i parametri che potete passarle sono due: il parametro *coordinata* ed il parametro *mappa*. Il parametro coordinata rappresenta la coordinata del punto che deve essere disegnato, mentre il parametro *mappa* definisce se il parametro *coordinata* è una coordinata x o una coordinata y e la direzione della trasformazione. I valori validi per il parametro *mappa* sono:

| Mappa | Valore restituito |
|---|---|
| 0 | Riporta la coordinata esterna x sulla coordinata fisica x. |
| 1 | Riporta la coordinata esterna y sulla coordinata fisica y. |
| 2 | Riporta la coordinata fisica x sulla coordinata esterna x. |
| 3 | Riporta la coordinata fisica y sulla coordinata esterna y. |

Come esempio dell'istruzione PMAP definiamo un sistema di coordinate esterne da (−100, −100) a (100,100); lo schermo così ottenuto è mostrato nella Figura 9.5.

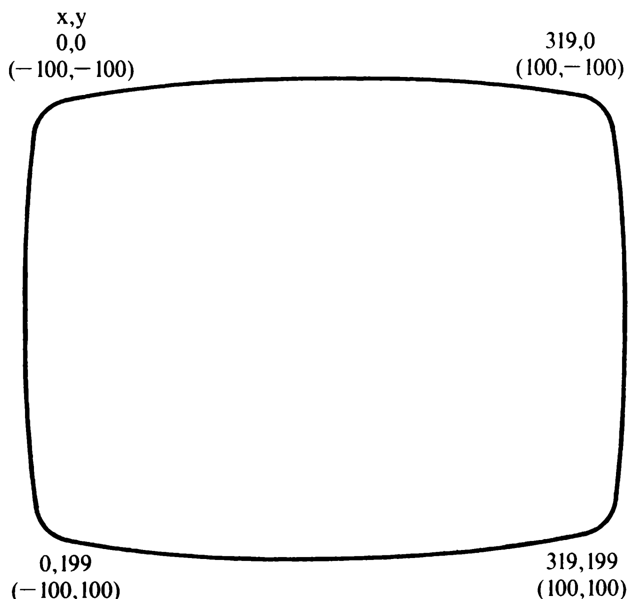



**Figura 9.5** Un sistema di coordinate esterne

Ora possiamo operare varie trasformazioni tra coordinate fisiche ed esterne nei diversi seguenti modi:

```
10 SCREEN 1:CLS
20 WINDOW SCREEN (-100,-100)-(100,100)
30 PRINT "FISICHE --> ESTERNE"
40 PRINT "  0 (x)        ";PMAP(0,2)
50 PRINT "  0 (y)        ";PMAP(0,3)
60 PRINT "319 (x)        ";PMAP(319,2)
70 PRINT "199 (y)        ";PMAP(199,3)
80 PRINT
90 PRINT "ESTERNE --> FISICHE"
100 PRINT "  0 (x)       ";PMAP(0,0)
110 PRINT "  0 (y)       ";PMAP(0,1)
120 PRINT " 50 (x)       ";PMAP(50,0)
130 PRINT "-50 (y)       ";PMAP(-50,0)
```

Ripetete il programma con diversi valori per WINDOW e PMAP. Potete

inoltre provare ad aggiungere un comando VIEW nella linea 15 per vedere come questo influisce sul risultato di PMAP.

## 9.11 Come determinare il valore di LRP: la funzione POINT

La maggior parte delle istruzioni grafiche utilizza come punto di partenza di default l'ultimo punto di riferimento (LRP); a volte, perciò, è importante essere in grado di determinare all'interno del programma, la posizione di LRP e questo può essere fatto per mezzo di una variazione della funzione POINT che abbiamo già incontrato all'inizio di questo capitolo. Quando la funzione è usata con un solo parametro, il BASIC considera che voi stiate richiedendo il valore corrente delle coordinate x e y di LRP, coordinate che possono essere ottenute sia nel sistema esterno che nel sistema fisico dello schermo. I parametri validi utilizzabili in questa versione di POINT sono i seguenti:

| Parametro | Valore restituito |
|---|---|
| 0 | Restituisce la corrente coordinata fisica x |
| 1 | Restituisce la corrente coordinata fisica y |
| 2 | Restituisce la corrente coordinata esterna x (fisiche=esterne se non è attiva WINDOW) |
| 3 | Restituisce la corrente coordinata esterna y (fisiche=esterne se non è attiva WINDOW) |

L'esempio seguente mostra l'uso della funzione POINT:

```
10 SCREEN 1:CLS
20 PRINT " LRP nella posizione (50,50) ESTERNO:"
30 LOCATE 5,1
40 PRINT "                        Fx  Fy  Ex  Ey"
50 PRINT :PRINT
70 PRINT "ESTERNE = FISICHE   :";
80 GOSUB 160
90 WINDOW SCREEN (-100,-100)-(100,100)
```

```
100 PRINT "WINDOW con SCREEN   :";
110 GOSUB 160
120 WINDOW (-100,-100)-(100,100)
130 PRINT "WINDOW senza SCREEN :";
140 GOSUB 160
150 END
160 LINE (10,10)-(50,50),0
170 FOR PT=0 TO 3
180 PRINT USING "####";POINT(PT)
190 NEXT PT
200 PRINT :PRINT :RETURN
```

e visualizza i valori:

```
 LRP nella posizione (50,50) ESTERNO:


                           Fx  Fy  Ex  Ey

ESTERNE = FISICHE    :    50  50  50  50

WINDOW con SCREEN    :   239 149  50  50

WINDOW senza SCREEN  :   239  50  50  50
```

Osservate il differente risultato quando la stessa istruzione WINDOW è utilizzata con o senza l'attributo SCREEN; ricorderete dalla precedente discussione sull'istruzione WINDOW che il tralasciare l'attributo SCREEN ha l'effetto di usare il sistema di coordinate cartesiane con il corretto orientamento, cioè con la componente y che diminuisce muovendosi dal basso verso l'alto lungo lo schermo, mentre la presenza di SCREEN fa sì che le coordinate siano orientate come lo sono normalmente, cioè con la componente y che aumenta scendendo lungo lo schermo.

## 9.12 Linguaggio per definizioni grafiche

Oltre alle istruzioni, peraltro molto versatili, discusse nei paragrafi precedenti, il PC supporta anche l'uso dei cosiddetti comandi del linguaggio per definizioni grafiche (*Graphics Definition Language*, GDL); sono, questi, comandi particolari, che servono a definire il modo in cui un oggetto dev'essere disegnato e sono contenuti in una normale stringa di caratteri. Quando l'Advanced BASIC esegue l'istruzione:

DRAW "*stringa*"

in realtà esamina il contenuto della stringa e interpreta i singoli caratteri, ciascuno rappresentante un comando al fine di disegnare l'oggetto nel modo richiesto. Gli oggetti vengono disegnati spostandosi da un pixel a quello adiacente e colorando ogni punto con uno dei quattro colori disponibili compresi nella tavolozza attiva. Ogni comando di movimento prende come punto di partenza l'ultimo punto di riferimento (*Last Referenced Point*, LRP).
I comandi GDL sono molti: permettono di scegliere la direzione e la distanza da percorrere con un movimento, il colore da usare, l'orientamento (angolo di rotazione) di un disegno, il fattore di scala (la dimensione del disegno, da 1/4 a 63.75 volte più grande). Tutti questi comandi sono seguiti da un argomento che può essere una costante o una variabile numerica indicante un valore per l'azione richiesta. Esiste infine un comando che permette di spostarsi dal punto corrente dello schermo ad una locazione relativa o assoluta.
Prima di esaminare i singoli comandi GDL e le loro funzioni, eseguiamo alcune prove dell'uso dei quattro più semplici: U, D, L e R (Up=su, Down=giù, Left=sinistra, Right=destra). Questo vi può fornire un assaggio dell'impiego dell'istruzione DRAW e dimostra anche alcune regole fondamentali a proposito dell'esecuzione delle stringhe di comandi GDL. Battete, perciò, ed eseguite questo programma:

```
10 SCREEN 1
20 CLS
30 D$="U 30"
40 DRAW D$
```

Come spiegato nel trattare l'istruzione SCREEN, quando si cancella lo schermo il valore di LRP viene posto uguale alle coordinate del centro dello schermo, cioè (160,100). L'istruzione DRAW, poi, traccia una linea dal centro dello schermo (LRP) fino al punto che si trova 30 pixel più in alto. LRP ora è l'ultimo punto disegnato, cioè la locazione (160,69).
Due sono le cose che dovete rilevare a questo punto: innanzitutto, ci sono 31 pixel illuminati, cioè il pixel di partenza più i trenta al di sopra; in secondo luogo, poiché non è stato indicato alcun colore, la linea è stata tracciata con l'ultimo colore richiesto da un qualsiasi comando grafico, eseguito anche prima che lo schermo fosse cancellato. Se non ci fossero stati comandi grafici precedenti, sarebbe stato impiegato il colore di default.
Aggiungete ora questa linea al programma:

```
50 DRAW "R   40;D30 L 40;"
```

Come potete vedere, la linea 50 inizia a disegnare da dove la linea 40 si era interrotta. Quest'ultima istruzione serve di esempio per altre regole: come prima osservazione, notiamo che l'istruzione DRAW ignora gli spazi bianchi all'interno della stringa; inoltre, un punto e virgola può essere posto dopo l'argomento che segue il comando GDL, ma è facoltativo se l'argomento è una costante; infine, vediamo come il programma disegni un quadrato colorando 30 punti verticalmente e 40 orizzontalmente, tenendo conto, cioè, del rapporto standard nella maggior parte degli schermi e cioè 4:3.
È importante ricordare che questi comandi GDL così versatili non sono influenzati dai comandi WINDOW e VIEW, ma usano solo il sistema completo delle coordinate fisiche dello schermo.

## I COMANDI GDL

Tenendo presenti questi concetti generali a proposito dell'istruzione DRAW, esaminiamo ora in questo paragrafo i comandi GDL uno per uno, che vengono raccolti nella Tabella 9.3.

### Comandi direzionali

I comandi direzionali sono otto: U, D, L, R, E, F, G, H; funzionano tutti nello stesso modo, specificando una direzione e muovendo il cursore della distanza indicata nell'argomento, sia esso una costante o una variabile numerica, che segue il comando. Il programma presentato precedentemente mostra l'uso dei primi quattro comandi direzionali con un argomento costante. Per comprendere il funzionamento degli altri quattro, modificate le linee 30 e 50 dello stesso programma, sostituendo E a U, F a R, G a D e H a L e poi eseguite il programma. Avete disegnato un rombo, e non un quadrato appoggiato su di un vertice: questo accade perché, anche se il rapporto dello schermo è 1:1, i comandi con direzione diagonale non tracciano linee con angoli di 45 gradi con la verticale (o l'orizzontale).
Mentre l'argomento *n* indica la distanza da percorrere in tutti e otto i comandi di direzione, l'effettivo numero di punti è *n* volte il fattore di scala, definito dal comando S.

### M - Movimento assoluto o relativo

Anche se i comandi elementari di movimento vi portano facilmente da un punto ad un altro vicino, disegnare una linea da un determinato punto ad

**Tabella 9.3** I comandi GDL

| | |
|---|---|
| U *n* | Muove il cursore verso l'alto |
| D *n* | Muove il cursore verso il basso |
| L *n* | Muove il cursore verso sinistra |
| R *n* | Muove il cursore verso destra |
| E *n* | Muove il cursore diagonalmente in alto a destra |
| F *n* | Muove il cursore diagonalmente in basso a destra |
| G *n* | Muove il cursore diagonalmente in basso a sinistra |
| H *n* | Muove il cursore diagonalmente in alto a sinistra |
| M *x,y* | Muove il cursore secondo coordinate assolute o relative. Se *x* è preceduto da un segno più (+) o meno (−), il movimento è relativo, altrimenti è assoluto |
| B | Muove il cursore ma non disegna punti |
| N | Muove il cursore, ma ritorna alla posizione originaria quando ha terminato |
| C *n* | Sceglie il colore *n*, con *n* da 0 a 3 in media risoluzione e da 0 a 1 in alta risoluzione |
| A *n* | Sceglie l'angolo *n*, con *n* che può variare da 0 a 3, indicando un angolo di *n*∗90 gradi |
| S *n* | Sceglie il fattore di scala, con *n* che varia da 1 a 255: il fattore di scala è *n*/4 |
| TA *n* | Ruota di un angolo *n*, con *n* che varia da −360 a +360 gradi. Gli angoli positivi indicano rotazione antioraria |
| P *colore, contorno* | Colora una figura: *colore* definisce il colore della figura e *contorno* il colore del bordo. Entrambi i parametri variano da 0 a 3 in media risoluzione e da 0 a 1 in alta risoluzione. |

un altro lontano e prefissato, non è per nulla facile; non è neppure facile raggiungere dalla posizione corrente (LRP) una locazione assoluta dello schermo. Per venirvi in aiuto, tra i comandi GDL ce n'è uno dal duplice scopo, il comando M.
Come esempio, battete ed eseguite il seguente programma:

```
10 SCREEN 1
20 CLS
30 D$="U30 R40 D30 L40"
40 DRAW D$
50 DRAW "M +40,-30"
```

Il comando M ha tracciato una linea obliqua da LRP al punto che sta 40 pixel a destra della coordinata *x* di LRP e 30 pixel più in alto della coordinata *y* di LRP. Un comando M usa coordinate relative quando il valore della *x* è preceduto dal segno più o dal segno meno.
Se la coordinata *x* del comando M invece, non è preceduta da alcun segno, viene eseguito un movimento assoluto. Aggiungete questa nuova linea al programma e provatelo:

```
35 DRAW "M 50,150"
```

Vediamo ora come M muova da LRP ad una posizione assoluta dello schermo: questo vi permette di stabilire il punto di partenza di un disegno (da disegnare con l'istruzione DRAW) in una determinata posizione — assoluta — dello schermo. Ci chiediamo, perciò, se non sia possibile rendere "vuota" la linea appena tracciata, piuttosto che disegnarla con il colore del fondo; ci serve, cioè un movimento "invisibile". Esiste un comando GDL proprio pensato a questo fine: il comando-prefisso B.

**B - Comando "muove senza disegnare"**

Questo comando dev'essere utilizzato come prefisso, cioè deve precedere ogni comando di movimento a cui si voglia riferirlo: il movimento avverrà in questo modo senza modificare alcun punto dello schermo. Inserite il prefisso B prima del comando M della linea 35 del programma precedente e osservate cosa accade.

**N - Comando "disegna e ritorna"**

Per disegnare una linea senza modificare l'ultimo punto di riferimento potete usare il prefisso N. Come per il precedente, anche questo dev'essere posto davanti ad ogni comando di movimento che si voglia influenzare. Fatelo con tutti i comandi di movimento dell'esempio appena fatto e poi eseguite di nuovo il programma. Cosa accade? Ogni movimento con prefisso viene eseguito, ma LRP rimane inalterato e perciò tutti i comandi iniziano a disegnare a partire dallo stesso punto. Sia il prefisso B che N influiscono solo sul movimento che li segue immediatamente.

**C - Sceglie il colore**

Il comando C definisce il colore per tutte le successive istruzioni di disegno, fino ad un nuovo comando C; è seguito da un argomento (costante o

variabile numerica) che indica il numero del colore desiderato. Nel modo grafico a media risoluzione, l'argomento può variare da 0 a 3, mentre in alta risoluzione sono permessi solo i valori 0 e 1. Un esempio di questo comando si trova nel seguente programma:

```
10 SCREEN 1
20 CLS
30 D$="U30 R40 D30 L40"
33 DRAW "C2"
35 DRAW"M50,150"
40 DRAW D$
```

**A - Definisce l'angolo di rotazione**

Il comando A serve per definire l'angolo di rotazione, cioè l'orientamento dei successivi comandi GDL. L'argomento, sempre costante o variabile numerica, che segue il comando A può variare da 0 a 3 in entrambi i modi grafici. All'inizio della stringa di comandi GDL per un dato disegno, il comando A ed il suo argomento indicano l'orientamento, con incrementi di 90 gradi, con cui il disegno dev'essere prodotto. Se trovate poco chiara questa spiegazione, questo veloce programma vi sarà di valido aiuto:

```
10 SCREEN 1:CLS
20 DRAW "A 0"
30 DRAW "U25 C2 NM-20,15 C1 M+20,15"
```

Osservate la direzione di questa figura; poi cambiate, nella linea 20, lo 0 con un 1 ed eseguite di nuovo il programma. Vedete come la variazione dell'argomento ha modificato la figura? Di nuovo modificate l'argomento con 2 e poi con 3 e riprovate il programma. Vi siete accorti che la figura ha sempre le stesse dimensioni, qualsiasi sia l'orientamento? In realtà le figure ruotate di 90 o 270 gradi subiscono una modificazione della scala, affinché appaiano identiche a quando sono disegnate con angoli di 0 o 180 gradi su uno schermo con rapporto standard di 4:3.

**TA - Rotazione**

Il comando TA ruota l'orientamento dello schermo del numero indicato di gradi per i successivi comandi GDL. L'argomento di TA (costante o variabile numerica) può variare da −360 a +360 gradi; un argomento positivo provoca una rotazione in senso antiorario del numero di gradi indicato, mentre un argomento negativo provoca una rotazione in senso orario.

L'argomento indica sempre un angolo di rotazione assoluto, nel sistema in cui l'angolo di 0 gradi, misurato a partire dall'orizzontale, è l'orientamento normale (di default). Il seguente programma vi mostra l'uso del comando TA con una variabile come argomento:

```
10 SCREEN 1:CLS
20 FOR X=10 TO 360 STEP 10
30 DRAW "TA=X;D40 L40 U40 R40"
40 NEXT X
```

**S - Definisce il fattore di scala**

Il comando S permette di ingrandire o ridurre le dimensioni di una figura di un quarto del valore dell'argomento (costante o variabile numerica) che segue il comando stesso. L'argomento può assumere valori compresi tra 1 e 255, in modo che i fattori di scala siano compresi tra 0.25 e 63.75. Il fattore di scala, moltiplicato per la distanza fornita dai comandi U, D, L, R, E, F, G, H ed M relativo, dà l'effettiva distanza percorsa dai comandi che seguono S. Il fattore di scala è valido finché non viene modificato da un successivo comando S. Provate ad aggiungere questa linea al programma precedente ed eseguitelo di nuovo:

```
15 DRAW "S 8"
```

La figura che ottenete è il doppio di quella originale.

**P - Colora gli oggetti**

Il sottocomando P del comando DRAW funziona in modo molto simile all'istruzione PAINT vista precedentemente in questo stesso capitolo. Anche P richiede due argomenti: il colore con cui riempire la figura ed il colore del contorno. Il parametro *colore* può variare da 0 a 3 nel modo Grafico a media risoluzione e da 0 a 1 nel modo Grafico ad alta risoluzione; mentre il parametro *contorno* può variare da 0 a 3 in entrambi i modi. Tutti e due gli argomenti devono sempre essere indicati nel comando P.

> NOTA: il comando P non possiede un'opzione di "pittura a motivi" come quella dell'istruzione PAINT.

Ecco un esempio per mostrare l'uso del comando P; osservate come il taglio delle linee funzioni con i comandi GDL esattamente come con tutte le altre istruzioni grafiche.

```
10 SCREEN  1:CLS
20 DRAW "U40 R200 D150 M160,100 BR3 P2,3"
```

### X - Esegue sottostringhe

Il comando X permette di eseguire una stringa di comandi GDL dall'interno di un'altra stringa GDL. La sottostringa è una comune stringa di comandi GDL, il cui nome deve seguire il comando X ed essere concluso con un punto e virgola. Questo è un comando molto utile da inserire in una stringa GDL, poiché permette di definire una parte del disegno una volta per tutte e poi di farla eseguire il numero di volte e nel momento richiesto. Analizzate il seguente programma per comprenderne meglio il funzionamento:

```
10 SCREEN 1:CLS
20 W$="R10D10L10U10BM+5,5NU5NL5ND5NR5BM-5,-5"
30 H$="D52R52U52L52M+26,-26M+26,26BL52"
40 DRAW H$+"BM+10,14 X W$;"+"BR21 X W$;"
```

## CARATTERISTICHE AVANZATE GDL

Se facciamo riferimento alla Tabella 9.3 che riporta tutti i comandi GDL, possiamo notare che in tutti i casi l'argomento *n* può essere una costante (come in quasi tutti i nostri esempi fin qui visti) o una variabile numerica. Il nome della variabile, se preceduto da un segno di = e seguito da un punto e virgola, è utilizzato come argomento dalla stringa GDL: ogni volta che questa viene eseguita, viene usato il valore corrente della variabile come valore di *n*. Questa caratteristica può essere sfruttata in molti modi diversi; vi proponiamo un semplice esempio di ciò che si può ottenere usando variabili come argomenti dei comandi GDL:

```
10 SCREEN 1:COLOR 1:KEY OFF:CLS
20 FOR X=1 TO 320
30 C=X MOD 3+1
40 A=X MOD 4
50 DRAW "C=C;A=A;U=X;"
60 FOR T=1 TO 100: NEXT
70 COLOR ,X MOD 2
80 NEXT
90 GOTO 10
```

## 9.13 Animazione grafica con GET e PUT

Nell'Advanced BASIC trovate due istruzioni, GET e PUT, che permettono di spostare immagini grafiche lungo tutto lo schermo con una velocità elevata. Nonostante siano molte le istruzioni BASIC, anche quelle riportate in questo capitolo, che vi consentono di spostare forme grafiche già definite, GET e PUT sono dotate di alcune caratteristiche che le rendono particolarmente adatte per l'animazione.
Pensiamo alle varie fasi necessarie per muovere una forma definita utilizzando le istruzioni PSET, PRESET e LINE: dovete innanzitutto visualizzare la figura, cancellarla rapidamente e poi rivisualizzarla nella nuova posizione sullo schermo. Questo processo, di solito, impiega un trentesimo di secondo, cioè il tempo di persistenza dell'immagine sullo schermo. Se dovete spostare pochi punti, anche le istruzioni PSET e PRESET possono rendere continuo il movimento, ma per oggetti più complessi il processo risulta troppo lungo e sconnesso per essere convincente.
Invece di disegnare le figure sullo schermo, le due istruzioni GET e PUT accedono direttamente alla memoria del video dell'interfaccia Colore/Grafica: GET estrae un'immagine dallo schermo e la immagazzina in un vettore numerico, mentre PUT pone il contenuto di un vettore nella memoria grafica.

### USO DELL'ISTRUZIONE GET

Per utilizzare l'istruzione GET dovete definire per mezzo delle coordinate degli angoli opposti del rettangolo un'area rettangolare di schermo, che verrà poi memorizzata. Potreste anche memorizzare l'intero schermo con un'unica istruzione GET, se fosse necessario, ma tenete conto che questo occuperebbe più di 16000 byte di memoria in entrambi i modi grafici.
GET usa un array numerico per immagazzinare l'immagine grafica: le dimensioni di questo array, naturalmente, dipendono da quanta parte dello schermo intendete memorizzare.

### OCCUPAZIONE DI MEMORIA

Prima di eseguire un'istruzione GET per la grafica di un programma, dovete riservare una parte di memoria, con l'istruzione DIM, in cui immagazzinare i dati. Le dimensioni del vettore dipendono dall'area di schermo e dal modo grafico in vigore quando viene eseguita l'istruzione GET. Di solito, conviene usare vettori di tipo intero per memorizzare i dati grafici, ma non sono esclusi vettori in precisione semplice o doppia.

Potete calcolare il numero di byte richiesti per contenere un'area di schermo utilizzando questa formula:

4+INT ((*colonne*∗(2/*modo*)+7) /8)∗*righe*

dove:

— *colonne* è il numero di colonne (larghezza dell'area)
— *modo* è il modo Grafico, 1 o 2
— *righe* è il numero di righe di schermo (altezza dell'area).

Ora che conoscete il numero di byte necessari, dividetelo per 2 se utilizzate un vettore intero, per 4 per un vettore in precisione semplice, per 8 per un vettore in doppia precisione: questo è il valore che dovete inserire nell'istruzione DIM.
Se vi serve calcolare spesso la dimensione dei vettori per l'istruzione GET, potete usare il seguente programma, che vi darà il numero di elementi necessari per un vettore intero:

```
20 CLS
30 INPUT "QUANTE COLONNE PER GET";C
40 INPUT "QUANTE RIGHE PER GET";R
50 IF R>200 THEN GOTO 40
60 INPUT "QUALE MODO --1 O 2";S
70 IF S<>1 AND S<>2 THEN GOTO 40
80 IF (S=1 AND C>320) OR (S=2 AND C>640) THEN GOTO
 30
90 ASIZE=(4+INT((C*(2/S)+7)/8)*R)/2
100 PRINT "NUMERO DI ELEMENTI INTERI NECESSARI:";A
SIZE
```

Per convertire la dimensione del vettore intero per un vettore in precisione semplice, dividete la risposta data da questo programma per 2, mentre per un vettore in doppia precisione, dividetela per 4.

## COME USARE GET E PUT PER L'ANIMAZIONE

La sintassi generale dell'istruzione GET è:

GET(*x1*,*y1*)−(*x2*,*y2*), *vettoreimmagine*

ove

— *x1* e *y1* sono le coordinate dell'angolo in alto a sinistra dell'area da memorizzare (GET)

— *x2* e *y2* sono le coordinate dell'angolo in basso a destra della stessa area
— *vettoreimmagine* è l'array numerico in cui saranno immagazzinate le immagini grafiche

Per il resto del capitolo, sarà importante capire a fondo due termini: un *vettoreimmagine* è l'array numerico che contiene tutte le informazioni riguardanti l'immagine e può essere creato solo da un'istruzione GET; un'area di schermo, invece, è quella parte di schermo in cui agiscono i comandi PUT e GET.
Il seguente programma usa queste due istruzioni nella loro forma più semplice per animare una figura di 8×8 pixel; prima di poter usare l'istruzione GET dovete avere sullo schermo il disegno che intendete muovere. Questo programma disegna la figura nell'angolo in alto a sinistra dello schermo e poi lo muove fino a quando non premete i tasti CTRL BREAK per interrompere il programma.

```
10 SCREEN 1:CLS
20 DIM F2%(10) 'Dimensiona il vettore immagine
29 REM DEFINISCE LA FIGURA
30 DATA 0,0,1,2,1,0,0,0
31 DATA 0,0,0,2,0,0,0,0
32 DATA 0,0,2,2,2,0,0,0
33 DATA 0,2,2,2,2,2,0,0
34 DATA 2,0,2,2,2,0,2,0
35 DATA 0,2,0,0,2,0,0,0
36 DATA 2,0,0,0,0,2,0,0
37 DATA 2,0,0,0,0,0,2,0
100 FOR ROW=0 TO 7
110 FOR COL=0 TO 7
120 READ PIXEL
130 PSET (COL,ROW),PIXEL
140 NEXT COL,ROW
150 GET (0,0)-(7,7),F2%  '"Legge" l'immagine
160 CLS
170 FOR ROW=0 TO 190     'Si sposta in basso
180 PUT (160,ROW),F2%    'Disegna l'immagine
190 FOR I=1 TO 20:NEXT I 'Ritardo prima di cancella
re
200 PUT (160,ROW),F2% 'Rimette l'immagine per cance
llarla
210 NEXT ROW
220 FOR COL=0 TO 310  'Si sposta da sin a dx
230 PUT (COL,100),F2%
240 FOR I=1 TO 20: NEXT I
250 PUT (COL,100),F2%
260 NEXT COL
270 GOTO 160  'Ripete il procedimento
```

Vale la pena di esaminare l'istruzione DIM nella linea 20: sostituendo il valore 8 sia per il numero di colonne che per il numero di righe nella formula che indica il fabbisogno di memoria, otteniamo:

```
4+INT((8*(2/1)+7)/8)*8=
4+INT((    23     )/8)*8=
4+INT(  2.875       )*8=
                    4+2*8=20
```

La dimensione dell'array è di 10, dal momento che il vettore è intero. Le istruzioni DATA, dalla linea 30 alla 37, definiscono la figura: ogni numero che compare in queste istruzioni è il numero di un colore (0 è il colore del fondo, mentre 1 e 2 sono i colori del testo).
Invece di una lunga serie di istruzioni PSET, queste istruzioni DATA definiscono la figura come una matrice, usando i numeri dei colori per ogni punto della figura di dimensione 8×8. Due loop FOR...NEXT, dalla linea 100 alla linea 140, leggono i numeri dei colori e con questi illuminano i pixel che si trovano nelle posizioni indicate dalle coordinate (COL,ROW). La linea 150 salva l'immagine così creata, cioè l'area di schermo delimitata dai due punti (0,0) (angolo in alto a sinistra) e (7,7) (angolo in basso a destra).
IL successivo loop FOR...NEXT, dalla linea 170 alla linea 210, muove la figura lungo lo schermo; osservate le coordinate dell'istruzione PUT nella linea 180: queste indicano il punto in cui deve apparire l'angolo in alto a sinistra del rettangolo salvato in memoria, mentre non sono più necessarie le coordinate dell'angolo in basso a destra. Il nome del *vettoreimmagine*, F2%, non deve essere racchiuso tra parentesi.
Nella linea 190 viene provocato un piccolo ritardo affinché l'immagine rimanga per un certo tempo sullo schermo; se questo loop venisse cancellato voi vedreste l'immagine lampeggiare mentre si sposta. Potete comprenderne il motivo osservando la linea 200, in cui l'immagine precedente viene cancellata: se non ci fosse alcun ritardo tra le due istruzioni PUT, queste verrebbero eseguite in un tempo molto minore di quello impiegato dal monitor per visualizzare il disegno, con il risultato di un'immagine che appare e scompare alternativamente.

## OPZIONI DELL'ISTRUZIONE PUT

L'aspetto in cui l'istruzione PUT si mostra più potente è la sua capacità di invertire, sovrapporre e combinare immagini sullo schermo. Qui descriveremo le opzioni di visualizzazione e mostreremo alcuni esempi di programmi che usano l'istruzione PUT. In ogni caso, vi renderete subito conto di quante altre siano le possibilità di animazione che sfruttano appieno questa istruzione in tutte le sue forme.

PUT disegna un'immagine sullo schermo, basandosi sul contenuto del vettore immagine e su ciò che già appare sullo schermo. Durante l'esecuzione di questa istruzione il colore di ogni pixel del vettore immagine interagisce con il colore del pixel corrispondente sullo schermo: i risultati possibili dopo questa interazione sono definiti da cinque diverse operazioni.
PUT infatti può agire con uno di questi tre operatori booleani: XOR (detto anche OR esclusivo), OR e AND; inoltre, può invertire l'immagine (PRESET) o operare una semplice operazione di trasferimento (PSET).
La sintassi dell'istruzione PUT con operatore è la seguente

PUT (*x*,*y*), *vettoreimmagine*, *operatore*

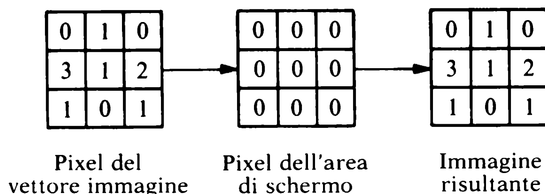
dove:

— *x* e *y* possono essere sia coordinate relative che assolute
— *vettoreimmagine* è il vettore numerico che contiene l'immagine grafica
— *operatore* è uno dei seguenti: XOR, OR, AND, NOT, PSET o PRESET.

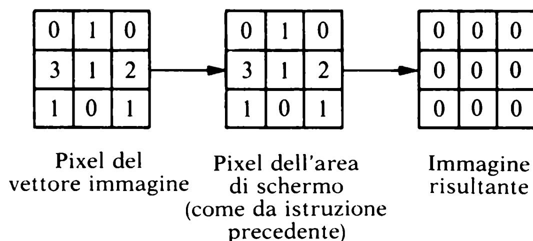
## L'OPERATORE XOR

Esaminiamo dapprima l'operatore XOR poiché è l'operatore di default per l'istruzione PUT. Con questa opzione, il valore di ogni pixel del vettore immagine viene confrontato con il valore del pixel corrispondente nell'area di schermo definita dalle coordinate che compaiono nell'istruzione PUT.
Se un pixel del vettore immagine ha lo stesso valore del corrispondente pixel sullo schermo, quest'ultimo prende il colore del fondo (0); al contrario, se il pixel del vettore immagine ha un colore di testo, cioè diverso da zero ed il pixel corrispondente sullo schermo ha il colore del fondo, quest'ultimo si cambia nel colore indicato nel vettore immagine.
Questo è il motivo per cui la prima di due istruzioni PUT identiche disegna l'immagine sullo schermo, mentre la seconda la cancella.

**Prima istruzione PUT**

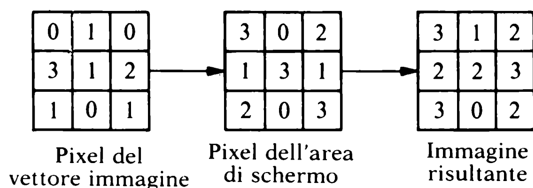


**Seconda istruzione PUT**

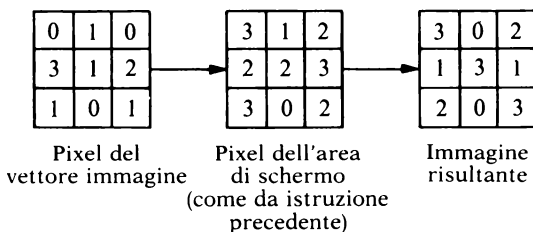



Nella Tabella 9.4 trovate tutte le possibili combinazioni e trasformazioni di colori prodotti dall'istruzione PUT con l'opzione XOR; osservate che queste operazioni creano alcuni interessanti colori quando un'immagine viene posta sopra un'altra già esistente: questo non è un fatto casuale, ma potete constatare che le trasformazioni sono pensate proprio al fine di lasciare intatte tutte le figure precedentemente disegnate.
Supponiamo di dover porre un'immagine in un'area di schermo in cui compaiono parti di fondo e parti di disegno in vari colori; ecco le trasformazioni che vengono prodotte:

**Prima istruzione PUT**




**Seconda istruzione PUT**




A mano a mano che l'immagine passa sopra diverse zone dello schermo in cui compaiono altri disegni, essa stessa cambia colore, ma i disegni

**Tabella 9.4** Trasformazioni di colori di PUT con XOR
(tra parentesi i colori della tavolozza 1)

| | | Pixel del vettore immagine | | | |
|---|---|---|---|---|---|
| | | **Fondo** | **Verde (Cyan)** | **Rosso (Magenta)** | **Marrone (Bianco)** |
| **Pixel sullo schermo** | **Fondo** | Fondo | Verde (Cyan) | Rosso (Magenta) | Marrone (Bianco) |
| | **Verde (Cyan)** | Verde (Cyan) | Fondo | Marrone (Bianco) | Rosso (Magenta) |
| | **Rosso (Magenta)** | Rosso (Magenta) | Marrone (Bianco) | Fondo | Verde (Cyan) |
| | **Marrone (Bianco)** | Marrone (Bianco) | Rosso (Magenta) | Verde (Cyan) | Fondo |

originali vengono esattamente reintegrati non appena l'immagine mobile viene cancellata.

## GLI OPERATORI OR E AND

Queste due operazioni logiche sono l'una l'opposto dell'altra: OR disegna un'immagine senza tener conto di cosa ci possa essere sotto di essa, mentre AND visualizza solo le parti di immagine che vanno a coprire un colore di testo.
Nella Tabella 9.5 sono riportate le trasformazioni causate dall'operatore OR, che hanno gli stessi risultati prodotti dall'operatore booleano OR. Se, per esempio, un pixel del vettore immagine ha un colore di valore 2 ed il corrispondente sullo schermo ha valore 1, l'istruzione PUT con operatore OR si comporta così:

| | **Decimale** | **Binario** |
|---|---|---|
| Pixel vettore immagine | 2 | 10 |
| Pixel schermo (OR) | 1 | 01 |
| Risultato | 3 | 11 |

In questo modo, PUT con l'operatore OR ha questi effetti (Tabella 9.5):

— Se il pixel del vettore immagine contiene il valore 0 (fondo), il pixel dello schermo rimarrà inalterato, qualsiasi sia il valore di partenza

Tabella 9.5 Trasformazioni di colore di PUT con OR
(tra parentesi i colori della tavolozza 1)

| | | Pixel del vettore immagine | | | |
|---|---|---|---|---|---|
| | | **Fondo** | **Verde (Cyan)** | **Rosso (Magenta)** | **Marrone (Bianco)** |
| **Pixel sullo schermo** | **Fondo** | Fondo | Verde (Cyan) | Rosso (Magenta) | Marrone (Bianco) |
| | **Verde (Cyan)** | Verde (Cyan) | Verde (Cyan) | Marrone (Bianco) | Marrone (Bianco) |
| | **Rosso (Magenta)** | Rosso (Magenta) | Marrone (Bianco) | Rosso (Magenta) | Marrone (Bianco) |
| | **Marrone (Bianco)** | Marrone (Bianco) | Marrone (Bianco) | Marrone (Bianco) | Marrone (Bianco) |

— Se il pixel del vettore immagine e quello dello schermo sono dello stesso colore, il pixel risultante è invariato
— Se il pixel del vettore immagine e quello dello schermo non sono dello stesso colore (e nessuno dei due ha valore 0), il colore risultante è il numero 3

Tabella 9.6 Trasformazioni di colore di PUT con AND
(tra parentesi i colori della tavolozza 1)

| | | Pixel del vettore immagine | | | |
|---|---|---|---|---|---|
| | | **Fondo** | **Verde (Cyan)** | **Rosso (Magenta)** | **Marrone (Bianco)** |
| **Pixel sullo schermo** | **Fondo** | Fondo | Fondo | Fondo | Fondo |
| | **Verde (Cyan)** | Fondo | Verde (Cyan) | Fondo | Verde (Cyan) |
| | **Rosso (Magenta)** | Fondo | Fondo | Rosso (Magenta) | Rosso (Magenta) |
| | **Marrone (Bianco)** | Fondo | Verde (Cyan) | Rosso (Magenta) | Marrone (Bianco) |

— Se l'immagine del vettore immagine viene visualizzata su di un'area di schermo completamente ricoperta dal colore del fondo, manterrà esattamente la propria forma e colore.

Anche AND agisce nello stesso modo dell'operatore booleano AND: nella Tabella 9.6 sono riportate le trasformazioni prodotte dall'uso, dell'istruzione PUT con l'operatore AND.

## LE OPERAZIONI PSET E PRESET

Anche queste due operazioni hanno effetto opposto: PUT con PSET trasferisce semplicemente l'immagine dal vettore all'area di schermo prescelta, eliminando tutto ciò che fosse eventualmente già presente in quell'area, mentre PRESET opera pure un trasferimento, ma invertendo i colori, cioè modificando i colori dei pixel del vettore immagine in questo modo:

| Colore del pixel del vettore immagine | Colore del pixel risultante sullo schermo |
|---|---|
| 0 | 3 |
| 1 | 2 |
| 2 | 1 |
| 3 | 0 |

Le operazioni PSET e PRESET, se usate in combinazione, producono rapidamente immagini negative dell'originale. Per convincervi di ciò, esaminate il programma presentato a pag. 319, che mostra come questa operazione sia di grande effetto nei giochi in cui è necessario "eliminare" qualcuno o qualcosa.
Per un'esemplificazione veloce dell'istruzione PUT con PSET e PRESET, battete ed eseguite il seguente programma: in questo caso il vettore immagine contiene tutto lo schermo e perciò tutti i colori presenti sullo schermo saranno invertiti.

```
10 DIM SCN1%(8004),SCN2%(8004)
20 CLS
30 SCREEN 1
40 COLOR 12,1  'Fondo rosso, tavolozza 1
50 FOR ROW=0 TO 190 STEP 11 'Per tutte le righe e le
colonne:
60 FOR COL=9 TO 319 STEP 21 'Definisce le coordinate
70 PSET (COL,ROW),0  'della stella
80 GOSUB 1000 'Disegna la stella
```

```
90 PAINT (COL,ROW+1),2,3 'La colora (PAINT) di magen
ta
100 NEXT COL
110 NEXT ROW
200 GET (0,0)-(319,199),SCN1%  'Memorizza l'immagine
210 PUT (0,0),SCN1%,PRESET 'Disegna l'immagine inver
sa
220 GET (0,0)-(319,199),SCN2% 'Memorizza l'immagine
inversa
230 PUT (0,0),SCN1%,PSET 'Disegna l'originale
240 PUT (0,0),SCN2%,PSET 'Disegna l'inversa
250 GOTO 230
1000 REM Subroutine per il disegno della stella
1010 LINE - STEP (3,3)
1020 LINE - STEP (6,0)
1030 LINE - STEP (-6,3)
1040 LINE - STEP (3,3)
1050 LINE - STEP (-6,-3)
1060 LINE - STEP (-6,3)
1070 LINE - STEP (3,-3)
1080 LINE - STEP (-6,-3)
1090 LINE - STEP (6,0)
1100 LINE - STEP (3,-3)
1110 RETURN
```

## COME ANIMARE NUMEROSE FIGURE

Il programma riportato nella Figura 9.6 definisce una figura di 8×8, che ha la vaga apparenza di un alieno, e un disco volante largo 32 pixel e alto 10; le istruzioni DATA contengono tutti i colori dei vari pixel per i due disegni. Innanzitutto il programma visualizza le due immagini e opera un'istruzione GET per memorizzarle in un vettore immagine.

```
1 REM ANIMAZIONE DI DUE FIGURE
2 KEY OFF
5 SCREEN 1,0:CLS
10 REM SEQUENZA DI ANIMAZIONE CON L'ISTRUZIONE GET
15 DEFINT F
20 DIM F1(8,8),F2(32,9),F3(12),F4(80)
29 REM ------ DEFINIZIONE FIGURA 1 ------
30 DATA 0,2,2,1,2,2,0,0
31 DATA 0,0,0,1,0,0,0,0
32 DATA 0,0,1,1,1,0,0,0
33 DATA 0,1,1,1,1,1,0,0
34 DATA 1,0,1,1,1,0,1,0
35 DATA 0,1,0,0,0,1,0,0
```

(continua)

```
36 DATA 1,0,0,0,0,0,1,0
37 DATA 1,0,0,0,0,0,1,0
39 REM ------ DEFINIZIONE FIGURA 2 ------
120 DATA 0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,2,2,2,2,0,0,0,
0,0,0,0,0,0,0,0,0,0
121 DATA 0,0,0,0,0,0,0,0,0,0,0,0,2,2,1,1,1,1,2,2,0,0,
0,0,0,0,0,0,0,0,0,0
122 DATA 0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,0,0,0,0,0,0,0,0
123 DATA 0,0,0,2,2,2,2,2,2,2,2,2,1,1,1,1,1,1,1,1,2,2,
2,2,2,2,2,2,2,0,0,0
125 DATA 0,0,2,2,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,1,2,2,0,0
126 DATA 2,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,1,2,2,2,2,2,2,2
127 DATA 2,2,2,2,2,2,2,2,2,2,1,2,2,2,2,2,2,2,2,2,1,2,
2,2,2,2,2,2,2,2,2,2
128 DATA 0,0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,1,1,2,2,2,2,
2,2,2,2,2,2,2,2,0,0
129 DATA 0,0,0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,2,2,2,2,2,0,0,0
130 DATA 0,0,0,0,0,0,0,0,2,2,2,2,2,2,2,2,2,2,2,2,2,2,
2,2,0,0,0,0,0,0,0,0
150 SCREEN 1
160 FOR Y=0 TO 7
170 FOR X=0 TO 7
180 READ F1(X,Y)
190 PSET(X,Y),F1(X,Y)
200 NEXT X,Y
210 GET (0,0)-(7,7),F3
300 CLS 'LEGGE IL VETTORE DEL DISCO VOLANTE
310 FOR Y=0 TO 9
320 FOR X=0 TO 31
330 READ F2(X,Y)
340 PSET(X,Y),F2(X,Y)
350 NEXT X,Y
360 GET (0,0)-(31,9),F4
365 CLS
370 FOR I=250 TO 70 STEP -1
380 PUT (I,100),F4
390 PUT (I,100),F4
400 NEXT I:PUT (I,100),F4
410 REM L'OMINO ESCE
420 FOR I=0 TO 40
430 PUT (I,190),F3
440 PUT (I,190),F3
450 NEXT I
460 LINE(44,189)-(86,110),3
470 REM ORA ESPLODE
480 FOR I=1 TO 10
```

(continua)

```
490 PUT (40,190),F3,PRESET
495 FOR J=1 TO 30: NEXT J
500 PUT (40,190),F3,PSET
505 FOR J=1 TO 30: NEXT  J
510 NEXT I
520 LINE(44,189)-(86,110),F3
525 PUT(40,190),F3
530 GOTO 530
```

**Figura 9.6** Animazione per mezzo delle istruzioni GET e PUT

Le linee dalla 370 alla 400 fanno volare il disco da destra a sinistra, mentre una successiva istruzione PUT (linea 400) ferma il disco volante in un punto dello schermo. Poi, nelle linee dalla 410 alla 450, l'alieno, di dimensione 8×8, avanza, un pixel per volta, dall'angolo in basso a sinistra per 40 pixel.
La linea 460 disegna una linea che parte dal disco e raggiunge l'ignaro alieno come un raggio trasportatore che lo inviti a salire sul disco. Entrano ora in funzione le istruzioni PUT che usano le operazioni PSET e PRESET e rapiscono la piccola figura: l'istruzione di linea 520 cancella il raggio, mentre l'istruzione di linea 525 cancella l'alieno.
Come abbiamo visto, GET e PUT hanno una grande varietà di applicazioni: se combinate, nel BASIC del PC, con altre istruzioni grafiche, vi offrono una vasta libreria di comandi e istruzioni con cui lavorare liberamente.

## 9.14 Come stampare una videata grafica

A volte è di vitale importanza ottenere una copia su carta del contenuto grafico di uno schermo: con l'aiuto del DOS potete stampare la vostra videata grafica su di una stampante grafica a matrice IBM o su una stampante Epson della serie MX con le opzioni grafiche.
Nel Capitolo 3 abbiamo esaminato il comando DOS GRAPHICS, che carica un programma residente nella memoria del PC. Questo programma permette, tra l'altro, di stampare il contenuto dello schermo. Dovete in questo caso iniziare la stampa con i tasti SHIFT PRTSC; se, prima di avviare il BASIC, date il comando GRAPHICS sotto DOS, potete stampare il contenuto di un qualsiasi schermo disegnato in BASIC.
Provate perciò ad inviare il comando A:GRAPHICS sotto DOS, dopo aver inserito nel drive A: il disco contenente il file GRAPHICS.COM; battete poi il comando A:BASICA per avviare l'Advanced BASIC e fate eseguire uno qualsiasi dei programmi di grafica presentati in questo capitolo. Dopo l'esecuzione del programma, premete i tasti SHIFT PRTSC e, se la vostra

stampante è accesa e pronta per l'uso, otterrete subito la copia dello schermo.
Se lo schermo è in modo Grafico a media risoluzione, i colori saranno resi con diverse gradazioni di grigio; lo schermo in modo Grafico ad alta risoluzione verrà stampato longitudinalmente sulla carta. Per copiare tutto lo schermo possono occorrere anche tre minuti.
Uno dei problemi che presenta la procedura qui descritta è che non potete copiare la videata senza includere almeno il prompt del BASIC che guasta l'effetto finale; infatti, quando un programma che disegna una figura sullo schermo termina l'esecuzione, il BASIC visualizza il prompt "Ok" sul disegno senza lasciarvi prima il tempo di copiarlo e questo perciò rimarrà anche sulla carta.
La subroutine presentata qui di seguito, che può essere inclusa in un qualsiasi programma BASIC, definisce un breve programma in linguaggio macchina che dà l'avvio della funzione di stampa, in modo che voi possiate iniziare a stampare prima che il programma sia terminato. Queta subroutine può essere chiamata dal programma tante volte quante necessarie, ed ogni istruzione GOSUB avrà come effetto la stampa della videata corrente.

```
500 PARRAY$="":RESTORE 560
510 FOR PARRLOOP=0 TO 4
520    READ V:PARRAY$=PARRAY$+CHR$(V)
525 NEXT PARRLOOP
530 PARRADDR=VARPTR(PARRAY$)+1
540 JUMPOFF=PEEK(PARRADDR)+PEEK(PARRADDR+1)*256
550 CALL JUMPOFF:RETURN
560 DATA &H55,&HCD,&H05,&H5D,&HCB
```

Per vedere il funzionamento di questa routine, utilizzate il programma di pag. 292: sostituite il loop infinito della linea 130 con un'istruzione LINE che utilizza l'opzione B per disegnare un contorno alla figura ed aggiungete una chiamata alla routine ora presentata, nella linea 140. Il programma risultante è questo:

```
10  SCREEN 1:COLOR 0,1
20  CLS
30  PSET (0,100)
40  FOR I=20 TO 320 STEP 20
50  LINE -(I,100+RND*10),2
60  NEXT I
65  MOTIVO1A$=CHR$(&H95)+CHR$(&H65)+CHR$(&H59)+CHR$(&
H56)
67  MOTIVO1B$=CHR$(&H56)+CHR$(&H59)+CHR$(&H65)+CHR$(&
H95)
70  PAINT (0,199),MOTIVO1A$+MOTIVO1B$,2
```

```
80  FOR I=1 TO 40
90  PSET (RND*319,RND*108)
100  NEXT I
110  CIRCLE (200,20),18,2
115  MOTIVO2$=CHR$(&HC3)+CHR$(&H3C)+CHR$(&H3C)+CHR$(&
HC3)
120  PAINT (200,20),MOTIVO2$,2
130 LINE (0,0)-(319,199),3,B
140 GOSUB 500:END
500 PARRAY$="":RESTORE 560
510 FOR PARRLOOP=0 TO 4
520    READ V:PARRAY$=PARRAY$+CHR$(V)
525 NEXT PARRLOOP
530 PARRADDR=VARPTR(PARRAY$)+1
540 JUMPOFF=PEEK(PARRADDR)+PEEK(PARRADDR+1)*256
550 CALL JUMPOFF:RETURN
560 DATA &H55,&HCD,&H05,&H5D,&HCB
```

Capitolo

# Il suono 10

Il suono può essere usato per dare un miglior aspetto ai programmi: potete inserire il rumore di un'astronave in avvicinamento in un programma di gioco o alcune note di esultanza per sottolineare una risposta esatta in un programma didattico. Inoltre il suono potrebbe diventare anche l'oggetto principale di un programma, dal momento che è relativamente semplice generare brani musicali.
Sono due i modi consentiti dal BASIC per creare dei suoni: l'istruzione SOUND e l'istruzione PLAY; le descriveremo entrambe in questo capitolo. Vi mostreremo come creare dei ritmi, come trascrivere per il PC musica già esistente, come sviluppare un programma musicale e come preparare una raccolta di effetti sonori.
Nella discussione delle due istruzioni SOUND e PLAY presumiamo una certa conoscenza dei concetti di base della musica (per esempio, cosa siano le note e le scale e come leggere alcune semplici notazioni musicali). Se non possedete queste informazioni, potreste far riferimento a qualche buon testo di musica o a qualche esperto nel campo.

## 10.1 L'istruzione SOUND

L'istruzione SOUND fa sì che il PC produca una nota o una pausa (cioè assenza di suono per un determinato intervallo di tempo). Nell'istruzione SOUND va specificato un numero che determina la *frequenza* della nota in cicli al secondo ed un altro che determina la *durata* della nota o della pausa. Provate le seguenti istruzioni SOUND in modo diretto così:

```
SOUND 37,10

SOUND 32767,1 ←——————— Questa genera una pausa

SOUND 2000,100

SOUND 20000,10
```

Il primo parametro dell'istruzione SOUND, quello che indica la frequenza della nota, deve essere un'espressione numerica compresa tra 37 e 32767 cicli al secondo (o Hertz); una frequenza di 32767 Hz crea un periodo di silenzio per la durata indicata.
L'intervallo di frequenze consentito dall'istruzione SOUND comprende ben più di qualche semplice beep, tipico dei computer: copre, infatti, l'intero campo di frequenze udibili dall'uomo. Nell'ultimo esempio, probabilmente, non udirete nient'altro che un leggero click ed un ronzio, poiché la frequenza indicata (20000 Hz) è troppo alta per essere percepita dalla maggior parte di noi.
Il secondo parametro indica la durata del suono in una particolare unità di misura che chiameremo "zic", dove 18.2 zic equivalgono ad un secondo; questa espressione numerica può variare tra 0 e 65535 zic, cioè tra 0 e all'incirca 60 minuti. Vedremo tra breve cosa significhi una durata di 0 secondi.
Assegnando i due parametri a variabili, manipolando queste opportunamente ed usando diverse istruzioni SOUND contemporaneamente, possiamo creare effetti di grande interesse; prima di descriverli, però, vediamo come il PC può controllare diverse istruzioni SOUND nello stesso programma.

## ISTRUZIONI SOUND MULTIPLE

Quando viene eseguita un'istruzione SOUND, il BASIC continua ugualmente il programma, anche durante il periodo in cui viene generata la nota. Di solito, se incontra un'altra istruzione SOUND, sospende l'esecuzione finché il suono precedente non è stato completato. Questo si osserva nel seguente esempio:

```
300 SOUND 1300,LTONO ←———— Viene eseguita questa istruzione, se-
310 ...                     guita dalle istruzioni dalla 310 alla
320 ...                     350, mentre viene suonata la nota
330 ...
340 ...
350 ...
```

```
360 SOUND 1200,LTONO
370 ...
380 ...
```

Se il primo suono non è terminato quando si raggiunge questa istruzione, il BASIC resta in attesa

Invece, se l'istruzione SOUND ha una durata 0, il suono corrente viene interrotto, come potete vedere:

```
300 SOUND 1300,LTONO
310 ...
320 ...
330 ...
340 ...
350 ...
360 SOUND 1200,0
370 ...
380 ...
```

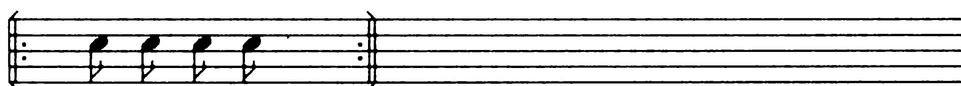
Se il primo suono è ancora in esecuzione, questa istruzione lo fa terminare ed il BASIC continua il programma

## TEMPO E RITMO

Combinando istruzioni SOUND che creano note con altre che creano pause, potete comporre motivi ritmici, del tipo di quello che segue:

```
SOUND 300,15:SOUND 32767,30:SOUND 400,36.4
```

Abbiamo generato una nota di 300 Hz per 0.8242 secondi (15 zic diviso per 18.2 zic al secondo = 0.8242 secondi), seguita da una pausa di 1.6484 secondi e da un'altra nota di 40 Hz per la durata di due secondi.
Nel PC il *tempo* di un ritmo è il numero di battiti per minuto (e non il numero di scansioni di una battuta musicale, rispetto all'intero, come viene definito in musica). In questa trattazione assegnamo sempre il valore di una semiminima (una nota da un quarto) ad ogni battito: perciò, il tempo risulta indicare il numero di note da un quarto per minuto.
Nella Tabella 10.1 sono elencate le conversioni principali per le durate delle note o delle pause da zic a secondi per un dato tempo. La tabella, cioè, ci dice ad esempio quanti zic deve durare l'istruzione SOUND se la nota è una croma (un ottavo) ed il tempo di 120 battiti al minuto.
Creiamo ora un semplice motivo ritmato con una successione di istruzioni SOUND. Ecco il ritmo che vogliamo produrre:



Tab. 10.1 Calcolo del tempo per l'istruzione SOUND

| Tempo (battiti al minuto) | Zic per una nota di un mezzo | Zic per una nota di un quarto | Zic per una nota di un ottavo puntata[1] | Zic per una nota di un ottavo |
|---|---|---|---|---|
| 60 | 36.40 | 18.20 | 13.65 | 9.10 |
| 80 | 27.30 | 13.65 | 10.24 | 6.83 |
| 100 | 21.84 | 10.92 | 8.19 | 5.46 |
| 120 | 18.20 | 9.10 | 6.83 | 4.55 |
| 140 | 15.60 | 7.80 | 5.85 | 3.90 |
| 160 | 13.65 | 6.83 | 5.12 | 3.41 |
| 180 | 12.13 | 6.07 | 4.55 | 3.03 |
| 200 | 10.92 | 5.46 | 4.10 | 2.73 |
| 220 | 9.93 | 4.96 | 3.72 | 2.48 |
| 240 | 9.10 | 4.55 | 3.41 | 2.28 |

1) Il punto dopo una nota o una pausa ne aumenta la durata della metà

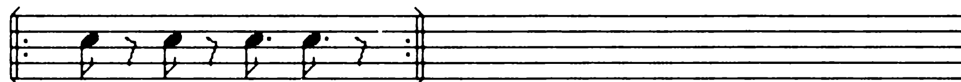
Queste sono semplicemente quattro note di un ottavo in sequenza: per distinguere ogni nota, assegneremo a ciascuna di esse, arbitrariamente, una diversa frequenza (utilizziamo la notazione musicale solo per il ritmo). Usiamo un tempo di 120 battiti al minuto: guardando la Tabella 10.1, vediamo che una nota di un ottavo a 120 battiti al minuto corrisponde a 4.55 zic per l'istruzione SOUND; ogni nota perciò può essere generata dalla seguente forma dell'istruzione:

```
SOUND FREQ,4.55
```

Per generare quattro note una dopo l'altra e poi ripetere l'intera sequenza, provate questo programma:

```
10 SOUND 300,4.55
20 SOUND 500,4.55
30 SOUND 700,4.55
40 SOUND 900,4.55
50 GOTO 10
```

Proviamo ora con un ritmo più complesso.



Questo è composto da una nota di un ottavo, seguita da una pausa, pure di un ottavo, poi da un'altra nota ed un'altra pausa entrambe di un otta-

vo, da due note puntate di un ottavo ed infine da una pausa finale, della stessa lunghezza delle precedenti. Di nuovo, proviamo a creare questo motivo ad un tempo di 120 battiti al minuto. Ricordiamo che per produrre una pausa la frequenza indicata nell'istruzione SOUND dev'essere di 32767. Ed ecco il programma che esegue questo motivo.

```
10 SOUND 100,4.55
20 SOUND 32767,4.55
30 SOUND 300,4.55
40 SOUND 32767,4.55
50 SOUND 500,6.83
60 SOUND 300,6.83
70 SOUND 32767,4.55
80 GOTO 10
```

### COME USARE L'ISTRUZIONE SOUND

L'istruzione SOUND, in combinazione con dei loop FOR...NEXT è molto utile per creare effetti sonori che rendono più piacevoli i vostri programmi. Provate il seguente programma:

```
10 FOR X=1000 TO 5000 STEP 200
20    SOUND X,2
30 NEXT X
40 GOTO 10
```

Potete anche utilizzare i loop annidati, come nel programma che segue:

```
 10 FOR Y=10 TO 200 STEP 20
 20   FOR A=100 TO 800 STEP 200
 30        SOUND Y+A,1
 40   NEXT A
 50   FOR B=100 TO 1500 STEP 200
 60        SOUND Y+B,1
 70   NEXT B
 80   FOR C=1000 TO 2000 STEP 200
 90        SOUND Y+C,1
100  NEXT C
110  FOR D=2000 TO 1200 STEP -200
120       SOUND Y+D,1
130  NEXT D
140 NEXT Y
150 GOTO 10
```

Per meglio vedere come un suono possa migliorare o sottolineare l'out-

put, fate eseguire questo programma, che richiede degli input e genera un suono, scelto tra due, ad indicare una risposta corretta o meno.

```
 10 CLS:PRINT "Qual e' la capitale della Danimarca?"
 20 PRINT:PRINT "1. Reykjavik"
 30 PRINT "2. Glouchester"
 40 PRINT "3. Copenhagen"
 50 PRINT "4. Goteborg"
 60 INPUT "Quale di queste ";A
 70 ON A GOTO 80, 80, 100, 80
 80 SOUND 50, 10
 90 GOTO 10
100 SOUND 300, 1:SOUND 400, 2:SOUND 900, 3
110 GOTO 10
```

Il suono prodotto da un'istruzione SOUND può anche essere utilizzato per fornire un segnale di avviso all'operatore. Supponiamo che il PC stia controllando un processo che non richiede un'attenzione costante: se il computer rileva qualcosa che va verificato immediatamente, il PC può saltare ad una routine che sia in grado di attirare l'attenzione dell'operatore.
La subroutine qui presentata può essere utilizzata per mettere sull'avviso un operatore non troppo distante, quando è richiesta la sua attenzione: la routine dapprima definisce come evento di interruzione la pressione di un tasto funzione nella linea 3080 in modo che l'operatore possa inviare segnali al PC. Il nucleo della routine, poi, sta tra le linee 3100 e 3170, che visualizzano un messaggio di richiamo, suonano una nota e poi cancellano il messaggio in modo da creare un output lampeggiante.

```
3060 ' **** ROUTINE DI AVVISO ALL'OPERATORE ****
3070 '
3080 KEY (1) ON:ON KEY (1) GOSUB 3180 ' Usa l'interce
ttamento per uscire dalla routine.
3090 CLS
3100 LOCATE 10,10:PRINT SPACE$(30) ' Cancella la prim
a riga del messaggio.
3110 FOR T=0 TO 200:NEXT T
3120 LOCATE 10,10
3130 PRINT "ATTENZIONE!! - E' NECESSARIO INTERVENIRE!
!"
3140 PRINT:PRINT TAB(10);"PREMERE IL TASTO DI FUNZION
E 1"
3150 SOUND 1000,10
3160 FOR T=0 TO 300:NEXT T
3170 GOTO 3100
3180 RETURN 3190 ' Questo in conseguenza del GOSUB ne
lla linea 3080.
```

```
3190 BEEP:RETURN ' Ritorna alla routine generale di c
ontrollo dell'errore.
```

Provate dapprima questa subroutine come programma a sé stante, modificando la linea 3190 in:

```
3190 BEEP:END
```

## 10.2 Esempi di effetti sonori

Questi programmi sono un ulteriore esempio dell'uso dell'istruzione SOUND per creare differenti effetti sonori; osservate in particolare che variando i parametri di SOUND si modifica il tipo di suono prodotto.

```
10 ' PROIETTILE
20 FOR I = 6000 TO 1000 STEP -20
30 SOUND I,.5
40 NEXT I
50 GOTO 20
```

```
10 ' RUBINETTO GOCCIOLANTE
20 I = 100
30 I = I + I^1.01
40 SOUND I,40/I
50 IF I>1000 THEN 60 ELSE GOTO 30
60 SOUND 32767,30 ' Ritardo di 30 zic.
70 GOTO 20
```

```
10 ' SIRENA
20 FOR I = 1000 TO 540 STEP -10
30       SOUND I,.5
40 NEXT I
50 FOR I = 540 TO 1000 STEP 10
60       SOUND I,.5
70 NEXT I
80 GOTO 20
```

## 10.3 L'istruzione PLAY

L'istruzione PLAY, propria dell'Advanced BASIC, permette di definire suoni con tecniche in qualche misura più vicine alla notazione musicale; potete, ad esempio, utilizzare le lettere A(La), B(Si), C(Do), D(Re), E(Mi), F(Fa) e G(Sol) per rappresentare le sette note musicali:

```
PLAY "DADA"
```

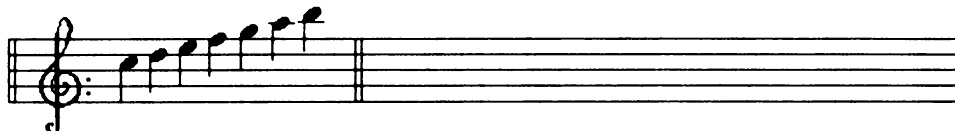
Questa istruzione esegue due volte le note Re e La.
La sintassi dell'istruzione PLAY è la seguente:

PLAY "*stringa*"

L'argomento *stringa* è il nucleo dell'istruzione: come nell'istruzione SOUND, questa stringa indica quali note o pause debbano essere suonate. Provate, per sincerarvene, questa istruzione:

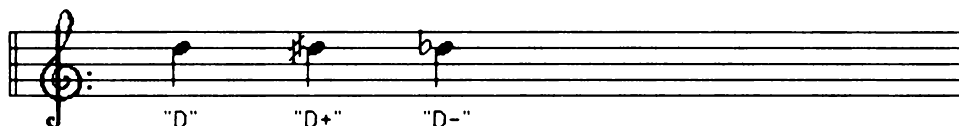
```
PLAY "CDEFGAB"
```

Questa è l'istruzione che esegue una scala in Do maggiore, qui riportata.

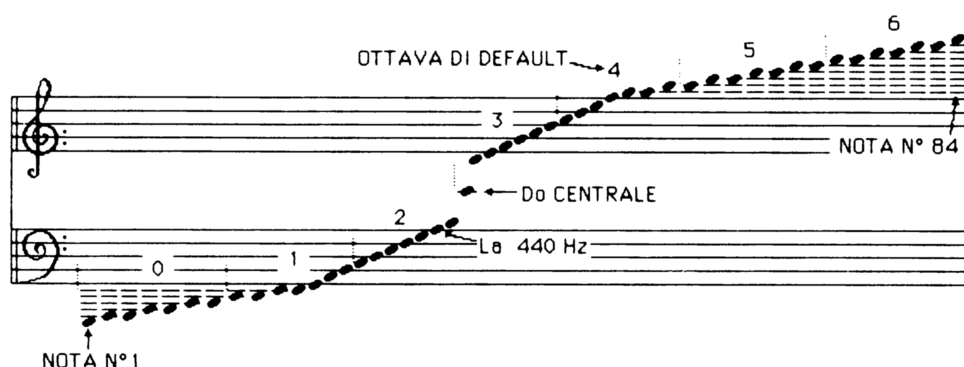


### COME INDICARE NOTE E OTTAVE

Ogni lettera da A a G, sia maiuscola che minuscola, inserita nella stringa dell'istruzione PLAY genera la nota corrispondente; se fate seguire la nota da un segno + o da un #, la nota risulterà aumentata di mezzo tono, cioè verrà eseguito il suo diesis, mentre un – dopo la nota esegue il bemolle, cioè la abbassa di mezzo tono.



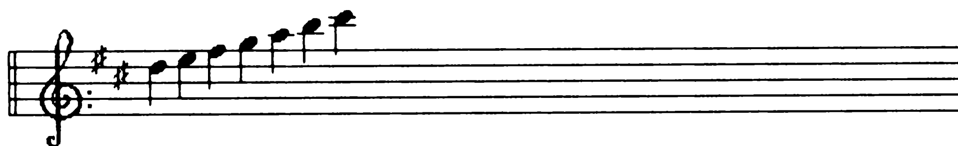
L'istruzione PLAY è in grado di eseguire un intervallo di 84 toni, che comprende sette ottave complete, numerate come segue:



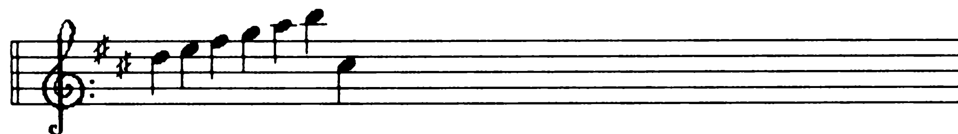

Se non specificate l'ottava, le note della stringa di PLAY saranno quelle dell'ottava numero 4; provate per esempio questa istruzione:

```
PLAY "DEF#GABC#"
```

Questa scala (la scala in Re maggiore), si abbassa di un'ottava quando viene suonato il Do diesis; infatti, invece di:



l'istruzione PLAY ha suonato questo:



Tutte le note di questo esempio fanno parte della stessa ottava e perciò, quando è la volta del Do diesis, l'istruzione PLAY scende fino all'inizio dell'ottava, cioè al Do diesis dell'ottava numero 4.
Per suonare una nota in un'altra qualsiasi ottava, dovete indicare il cambiamento; sono possibili due forme di cambiamento di ottava nell'istru-

zione PLAY, una relativa e l'altra assoluta. La forma relativa usa i simboli > (maggiore) e < (minore) come parte della stringa: il > indica "sposta l'ottava corrente in alto di uno", mentre < indica "abbassa l'ottava corrente di uno". Se l'ottava corrente è quella di default, cioè la numero 4, il comando > fa in modo che le note successive siano prese dall'ottava numero 5; la nuova ottava indicata rimane in vigore fino al successivo esplicito cambiamento. Provate questa dimostrazione pratica del cambiamento relativo di ottave:

```
10 PLAY "DEF#GAB>C#DEF#GAB>C#D"
20 PLAY "DC#<BAGF#EDC#<BAGF#ED"
```

Non si ottiene alcuna modifica dell'ottava da un comando > quando l'ottava corrente sia la 6, come dal comando < quando l'ottava corrente sia la numero 0.
La forma assoluta del cambiamento di ottava utilizza lo specificatore di ottava, O, seguito dal numero dell'ottava scelta, da 0 a 6. Un esempio:

```
PLAY "O3ADAD"
```

Questa istruzione suona le note La, Re, La e ancora Re dell'ottava numero 3. Potreste anche usare questa sintassi:

PLAY "O=*variabile;stringa*"

Provate il seguente programma per vedere come muovere una scala (in Do maggiore) su e giù di un'ottava per volta usando lo specificatore di ottave:

```
10 FOR OTTAVA=0 TO 6
20 PLAY "O=OTTAVA;CDEFGAB"
30 NEXT OTTAVA
40 GOTO 10
```

Dal momento che il numero dell'ottava cambia ad ogni Do dell'intervallo consentito all'istruzione PLAY, potrebbe risultare necessario utilizzare lo specificatore di ottave in diversi punti della stringa: per esempio, per suonare correttamente la scala di Re maggiore di uno degli esempi precedenti, usate questa istruzione:

```
PLAY "DEF#GABO5C#"
```

Le prime sei note di questa stringa (Re, Mi, Fa diesis, Sol, La e Si) vengono suonate nell'ottava di default, la numero 4, mentre per suonare il Do diesis superiore dovete passare all'ottava numero 5.

Tutte le note che seguono uno specificatore di ottava apparterranno a quell'ottava finché non si incontra un diverso specificatore. Provate a far eseguire questa istruzione:

```
PLAY "DGEDO1GGEO5A#DF#"
```

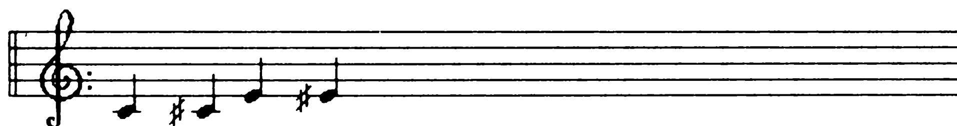
Le prime quattro note –Re, Sol, Mi e Re– vengono suonate nell'ottava di default; le tre successive –Sol, Sol e Mi– nell'ottava numero 1, mentre le ultime tre –La diesis, Re e Fa diesis– nell'ottava numero 5.
Riportate ora il cursore all'inizio dell'istruzione e premete ENTER: osservate che le prime quattro note vengono questa volta suonate nell'ottava numero 5, perché l'ultima istruzione PLAY eseguita aveva posto come ottava corrente proprio questa.

## COME USARE I NUMERI AL POSTO DELLE NOTE

Un altro modo di indicare le note è di usare i numeri da 1 a 84: in questo caso dovete far precedere i numeri dalla lettera N (maiuscola o minuscola, non importa), come in questo esempio:

```
PLAY "N37N38N39N40"
```

Questa istruzione suona le seguenti note:



Il Do centrale corrisponde alla nota numero 37:




Quando indicate le note con i numeri invece che con le lettere, non dovete indicare l'ottava: infatti, ogni numero da 1 ad 84 si riferisce ad una specifica altezza. L'incremento di uno equivale infatti a un semitono.
Un numero uguale a 0 non indica una nota ma una pausa; osservate come agisce una pausa con questa istruzione:

```
PLAY "N37NON39NON41NON42"
```

che separa ogni nota dalla successiva con una pausa di durata uguale a quella delle note:



I numeri delle note possono essere sostituiti da variabili, come nell'istruzione che segue:

```
PITCH=37
PLAY "N=PITCH;"
```
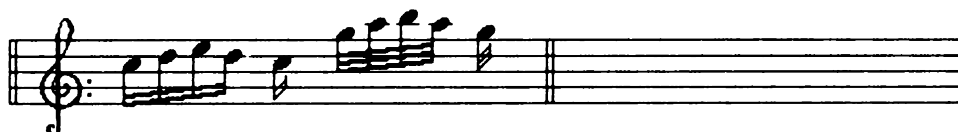
Infatti, potete usare la sintassi N=*variabile*; con tutti i parametri contenuti nella stringa dell'istruzione PLAY.

## COME INDICARE LA DURATA DI UNA NOTA

Finora, tutte le note suonate dall'istruzione PLAY avevano la stessa durata, ma è possibile creare anche note di durata diversa.
Potete variare la durata di ogni nota usando lo specificatore di lunghezza L all'interno della stringa, come in questo caso:

```
PLAY "L16CDEDCL32GABAG"
```

in cui la prima L, seguita dal numero 16, fa sì che le note successive (Si, Re, Mi, Re ed infine Do) siano delle semicrome (dei sedicesimi) e ciò dura fino a che l'istruzione PLAY trova nella stringa un altro specificatore di lunghezza, cioè L32: le ultime note (Sol, La, Si, La, Sol) sono dei trentaduesimi.
L'effetto finale è questo:

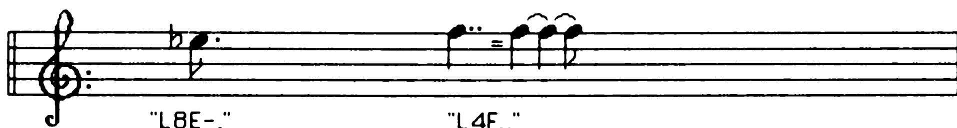


Il parametro di lunghezza, *n*, che segue lo specificatore L, può essere un numero o una variabile, compresi entrambi tra 1 e 64; la lunghezza di de-

fault è 1. Dopo che è stata indicata una lunghezza *n*, tutte le note successive, finché non venga indicato diversamente, vengono suonate con una durata di 1/*n*: L64 perciò indica una nota di un sessantaquattresimo.
Un altro modo di stabilire la lunghezza delle note è di far seguire ogni nota da un numero: in questo caso, però, la lunghezza indicata influisce su quella nota e non sulle successive. Un esempio:

```
PLAY "L4G16A8BB"
```

In questa istruzione L4 definisce la lunghezza di default per la stringa ad un quarto; le prime due note, però, sono seguite dall'indicazione della propria durata, che sovrasta quella di default (G16 indica un Sol di durata di un sedicesimo, mentre A8 indica un La di durata di un ottavo). La durata delle due ultime note (Si e Si), comunque, è ancora di un quarto, come era stato stabilito all'inizio della stringa.
Aggiungendo un punto dopo una nota, questa risulta di una volta e mezzo la propria durata: più di un punto fa sì che la nota duri $(3/2)^n$ volte il tempo originario, con *n* numero dei punti. Per esempio:




## COME STABILIRE IL TEMPO

Potete inserire anche il comando T nella stringa dell'istruzione PLAY ad indicare il tempo dell'esecuzione. Ricordate che il tempo si riferisce al numero di note da un quarto per minuto e perciò determina quanto veloce (o quanto lento) debba essere suonato un brano musicale.
L'argomento che segue il comando T indica il numero di battiti da un quarto al minuto e può variare da 32 a 255. Se non ne indicate esplicitamente un altro, il tempo di default è di 120 battiti al minuto.
Provate queste due istruzioni PLAY per vedere l'effetto delle variazioni di tempo:

```
PLAY "L8DGCFB-E-A-D-G-BEA"
PLAY "T250L8DGCFB-E-A-D-G-BEA"
```

## COME INDICARE LE PAUSE

Potete indicare la lunghezza di una pausa proprio come indicate quella di una nota: usando la lettera P, per pausa, otterrete un silenzio della durata indicata dall'argomento *n*; come prima, *n* può variare tra 1 e 64 e la lunghezza della pausa è di 1/*n*. Provate con questa istruzione:

```
PLAY "L64CP64CP64L16CP16CP16L8CP8CP8"
```

In questo caso, il primo comando L64 definisce la lunghezza delle note ad un sessantaquattresimo e questo vale per i primi due Do; entrambe queste note sono separate dalla successiva da una pausa di un sessantaquattresimo, come indicato dai due P64. La notazione musicale equivalente alla stringa è questa:

Potete anche specificare la lunghezza delle pause con delle variabili, in questo modo:

```
PLAY "L4DP=PAUSA;DP=PAUSA;D"
```

Questa istruzione suona tre Re di un quarto l'uno, separati da pause la cui lunghezza è determinata dal valore della variabile PAUSA, cioè è uguale a 1/PAUSA battiti. Se perciò PAUSA vale 8, l'istruzione equivale a questa notazione musicale:

Il punto come specificatore è utilizzabile anche con le pause.

## L'ARTICOLAZIONE DELLE NOTE

Le note dell'istruzione PLAY possono essere suonate con tre differenti articolazioni: Normale, Legato, Staccato. Con articolazione si intende la parte della lunghezza indicata per una nota che viene effettivamente suonata.

Il Legato, indicato con ML (*Music Legato*), fa sì che le note vengano suonate per l'intera durata specificata: questo produce una melodia continua e inoltre le note consecutive uguali vengono suonate come se fossero una sola. Un esempio:

```
PLAY "MLL4T200FFGGAA"
```

Il modo Normale, indicato con MN (*Music Normal*) è l'articolazione di default e fa sì che le note vengano suonate per 7/8 della loro lunghezza: questo pone maggior enfasi su ogni singola nota, come potete vedere confrontando questa esecuzione con la precedente:

```
PLAY "MNL4T200FFGGAA"
```

Lo Staccato, indicato con MS (*Music Staccato*), fa in modo che le note che seguono siano suonate per 3/4 della loro lunghezza: questo pone un accento ancora più marcato sulle singole note; ecco l'istruzione da confrontare con le altre:

```
PLAY "MSL4T200FFGGAA"
```

Un parametro di articolazione rimane in effetto per tutte le note che lo seguono fino a che non ne viene indicato esplicitamente un altro; provate anche questa istruzione per vedere meglio la differenza tra i tre tipi di espressione:

```
PLAY "L4T200MLGGBMNGGBMSGGB"
```

In questo caso le prime tre note, della durata di un quarto, sono legate, in modo che i due Sol siano suonati come se fossero un'unica nota; le tre successive sono suonate normalmente, in modo che tutte le note siano separate ed infine le ultime tre sono staccate, in modo che la separazione risalti ulteriormente.

## SOLISTA E ACCOMPAGNAMENTO

L'istruzione PLAY può essere eseguita in due modi: Solista (*Music Foreground*, MF nella stringa di comandi) e Accompagnamento (*Music Background*, MB nella stringa). Se viene indicato MF nella stringa di un'istruzione PLAY, una qualsiasi successiva istruzione non viene eseguita finché non è stata suonata tutta la stringa di musica; se invece si è specificato MB, la successiva istruzione BASIC verrà eseguita senza attendere il termine dell'esecuzione musicale.

Per apprezzare le differenze tra Accompagnamento e Solista, provate questi due programmi:

```
10 N=1
20 PLAY "MFT100L4ADGDADGDA"
30 PRINT "Questa è l'esecuzione numero " N
40 N=N+1
50 GOTO 20
```

```
10 N=1
20 PLAY "MBT100L4ADGDADGDA"
30 PRINT "Questa è l'esecuzione numero " N
40 N=N+1
50 GOTO 20
```

Osservate la differenza nella visualizzazione del messaggio rispetto all'esecuzione del suono, nei due casi, MB ed MF.

## 10.4 Il modo Accompagnamento

Il vantaggio principale del modo MB, cioè Accompagnamento, sta nel fatto che un programma può dare l'avvio ad una stringa di note musicali, nel buffer di accompagnamento e poi continuare a svolgere altri compiti contemporaneamente alla musica: l'utilità delle capacità musicali del computer è così enormemente aumentata.
Spesso è essenziale che il programma applicativo sia in grado di controllare il progredire delle note nel buffer: se questo è vuoto, o quasi, è il momento di fornirgli altre note; un programma potrebbe inoltre avviare un brano di musica, eseguire altri lavori e poi dover attendere finché non sia stato suonato un passo particolare prima di riprendere l'attività. Quello che si richiede, cioè, è un modo per sincronizzare l'avanzamento delle note nel buffer con quello del programma applicativo.

### LA FUNZIONE PLAY

Per operare questo sincronismo con la musica suonata come accompagnamento, il BASIC fornisce la funzione PLAY (ben diversa dall'istruzione PLAY appena descritta). La sintassi della funzione è:

*n* = PLAY(*esprnum*)

dove *esprnum* è una variabile fittizia; la funzione restituisce il numero di note in attesa nel buffer di accompagnamento; *n* non varrà mai più di 32, poiché questa è la dimensione massima del buffer. L'azione intrapresa in funzione del valore della funzione PLAY dipende dal programma in cui essa è inserita.

## L'ISTRUZIONE ON PLAY

Un altro comando di grande utilità, l'istruzione ON PLAY, rende più facile ad un programma il compito di provvedere ad un accompagnamento musicale continuo con un disturbo minimo sull'esecuzione del resto del programma. Come abbiamo già visto nel paragrafo 6.4, la sintassi di ON PLAY è:

ON PLAY(*n*) GOSUB *linea*

dove *n* è un numero tra 1 e 32 che indica il numero di note che devono essere rimaste nel buffer per dare origine all'intercettamento; il parametro *linea* è il primo numero di linea della routine di intercettamento dell'evento PLAY.
Non appena il buffer contiene una nota meno di quanto indicato in *n*, si verifica l'evento e viene eseguita la routine chiamata. Un evento PLAY può verificarsi solo quando l'istruzione PLAY suona in modo MB; infatti il buffer non deve essere vuoto quando viene eseguita l'istruzione ON PLAY, altrimenti non parte alcun intercettamento.
La routine qui presentata può essere inserita con buoni risultati in un programma che esegue, occasionalmente, lunghi periodi di calcolo: se questo programma richiede input da parte dell'utente, il sottofondo musicale può rassicurarlo sul fatto che il programma non si sia interrotto:

```
10 WIDTH 40:SCREEN 0
20 ON PLAY(1) GOSUB 150
30 ' Inizia il lavoro. PLAY non e'
40 ' ancora attivato.
50 ' ....
60 ' ....
70 ' Tempo per il calcolo
80 GOSUB 150  ' Inizia il sottofondo
90 PLAY ON
100 ' Ancora calcolo
110 FOR X=1 TO 15 : COLOR X
120     PRINT "STO CALCOLANDO...";
125 NEXT X
130 GOTO 110
140 '
```

```
150 'subroutine ON PLAY
160 '
170 CLS: COLOR X+1:X=(X+1) MOD 15
180 PRINT "NON HO ANCORA FINITO !!"
190 PLAY "MBT120L8O1MSCFP8AFP8CFP8AFP4"
200 PLAY "MBO1FFP8GAGFEFP8GL1FP1P1P1P1"
210 RETURN
```

## USO DI PIÙ STRINGHE NELL'ISTRUZIONE PLAY

Fino ad ora abbiamo utilizzato l'istruzione PLAY con una sola stringa come argomento; sono molte, però, la maniere per introdurre più di una stringa nella stessa istruzione PLAY. Innanzitutto, potete usare l'operatore di concatenazione di stringhe, +, per collegare tra loro stringhe diverse. Per esempio:

```
A$="A"

B$="B"

PLAY "T160L8" + A$ + B$ + A$ + B$
```

Un altro modo è quello di usare l'operatore X (esecuzione di sottostringhe): quando questo viene inserito nella stringa di un'istruzione PLAY, la sottostringa indicata viene a far parte della stringa principale. Un esempio:

```
A$="A"

B$="B"

PLAY "T160L8XA$;XB$;XA$;XB$;"
```

Questa istruzione ha lo stesso effetto della seguente:

```
PLAY "T160L8ABAB"
```

La possibilità di far eseguire delle sottostringhe dall'istruzione PLAY vi consente di ripetere brani di musica in componimenti più ampi senza doverli riscrivere ogni volta; un ritornello potrebbe essere codificato in una sottostringa e poi eseguito ogni volta che sia necessario.
Le sottostringhe possono anche essere annidate, in questo modo:

```
A$="O4GB-AA"

B$="O5CO4B-AB-"
```

```
C$="O2DFEDXA$;XA$;"

PLAY "T100L16XC$;XA$;XB$;"
```

Quest'ultima istruzione PLAY equivale a quest'altra:

```
PLAY "T100L16O2DFEDO4GB-AAO4GB-AAO4GB-A
     AO5CO4B-AB-"
```
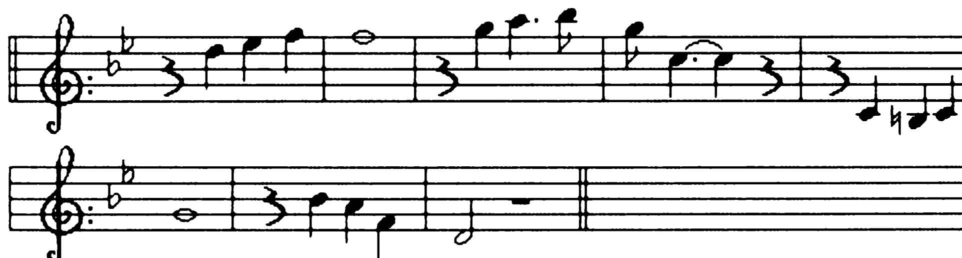
Ed ecco un programma che mostra l'uso delle sottostringhe: suona un accompagnamento di basso di un motivo jazz che sottolinea l'armonia del pezzo. La forma del motivo è AABA, cioè la prima parte è ripetuta due volte, seguita da una seconda parte (ritornello) e di nuovo dalla prima.

```
10 FRASEA$="FACAEB-AGDFGBCE-FE-B-AGDACDF#GBFDGECB-"
20 RITORNELLO$="CE-B-CGE-FE-B-DAB-FDGFE-G-B-G-A-CE-CD
-FA-FGB-CE"
30 PLAY "MLT120L8OOXFRASEA$;XFRASEA$;XRITORNELLO$;XFR
ASEA$;"
40 GOTO 30
```

## COME PASSARE DA UNO SPARTITO AL PC

Cerchiamo di utilizzare tutto quanto abbiamo imparato finora, per trascrivere un semplice motivo in modo che il PC possa suonarlo. La canzone che abbiamo scelto come prova è la seguente, dal titolo: "I left my heart in San Francisco":

"I LEFT MY HEART IN SAN FRANCISCO"



La prima cosa da osservare è il tempo del pezzo: la musica è scritta in 4/4, cioè in ogni battuta ci sono quattro note da un quarto, con un quarto per ogni battito; il tempo è di 120, cioè ci sono 120 battiti al minuto. Dovremo perciò iniziare la stringa con le notazioni:

```
MNT120L4
```

MN assicura la condizione normale; T120 pone il tempo a 120 battiti al minuto, mentre L4 viene scelta perché la maggior parte delle note del pezzo sono quarti e perciò avere questo valore come default rende più agevole la notazione.
Il prossimo passo è di guardare la chiave di tonalità in cui è scritto il pezzo, indicata in questo caso dai due bemolle all'inizio di ogni rigo: questi indicano che il pezzo è in Si bemolle e cioè che, se non indicato altrimenti, al posto di ogni Si e di ogni Mi che compaiono nel brano devono essere suonati i loro bemolle.
Pensiamo ora quale sia l'ottava in cui suonare: ricordiamo che il Do centrale, quello appena sotto il rigo in chiave di violino, è l'inizio dell'ottava numero 3; il Do superiore a questo, perciò, farà parte dell'ottava numero 4. Il motivo inizia proprio in questa ottava e così anche la stringa deve partire da questa; ma l'ottava 4 è quella di default e ciò ci permette di non aggiungere nient'altro alla stringa. L'ottava, comunque, dovrà essere modificata nella quinta battuta, come vedremo tra breve.
Ora codifichiamo il motivo, una nota (o pausa) per volta: per rendere tutto più semplice, scriviamo dapprima il motivo senza indicazioni di lunghezza, perché queste non influiscono sulla struttura della stringa.

P D E- G F1 P G AA8 B-8 G8 CC8C P O3C O2B O3C
ML ML
(Music Legato)

G1 P B- A F D2 P2

Nell'illustrazione precedente è stata scritta la parte di stringa corrispondente sotto ogni nota ed ogni pausa. Alcuni passaggi richiedono una speciale attenzione, in particolare il modo in cui sono resi il La puntato ed i Do legati: le due note puntate, La e Do, sono state scritte come un quarto seguito da un ottavo che vogliamo siano suonati insieme; per questo sono preceduti dall'indicazione di ML (Legato).
Osservate nell'illustrazione che il cambiamento di ottava usa lo specifica-

tore assoluto O. Potremmo sostituirlo con gli specificatori relativi < e >. Come mostrato, la musica inizia nella quarta ottava: il primo cambiamento, O3, può essere ottenuto con il comando relativo <; nello stesso modo, il seguente O2 richiede un altro comando di <, mentre un altro comando O3 a questo punto va sostituito con un >. Modificando in questo modo le variazioni di ottava, possiamo predisporre il programma in modo che l'utente decida da quale ottava far partire il motivo.
Dopo aver definito tutte le note e tutti i parametri necessari, possiamo raggrupparli in questa maniera:

```
10 MUSICA1$="MNT120L4O=X!;F4DE-GF1F4"
20 MUSICA2$="GMLAA8B-8G8C2CF2"
30 MUSICA3$="<C<B>CG1F4B-AA8F8D2D"
40 INPUT "Inserire l'ottava (da 0 a 6)",X!
50 IF (X!<0) OR (X!>6) THEN 40
60 PLAY MUSICA1$+MUSICA2$+MUSICA3$
70 GOTO 40
```

Provate questo programma con diverse ottave iniziali: osservate cosa accade quando partite dalle ottave 0 o 1 (la causa dell'effetto che così si ottiene è che nell'ottava 0 il comando < non ha significato). Provate anche a modificare altri parametri del programma: il tempo o la durata di alcune note; cercate, insomma, di ottenere l'interpretazione del motivo che più preferite.
Come esempio più complesso, proviamo a codificare una versione più sofisticata dell'accompagnamento di basso presentato prima.
Vediamo come "tradurre" le prime otto battute del pezzo: tanto per inco-
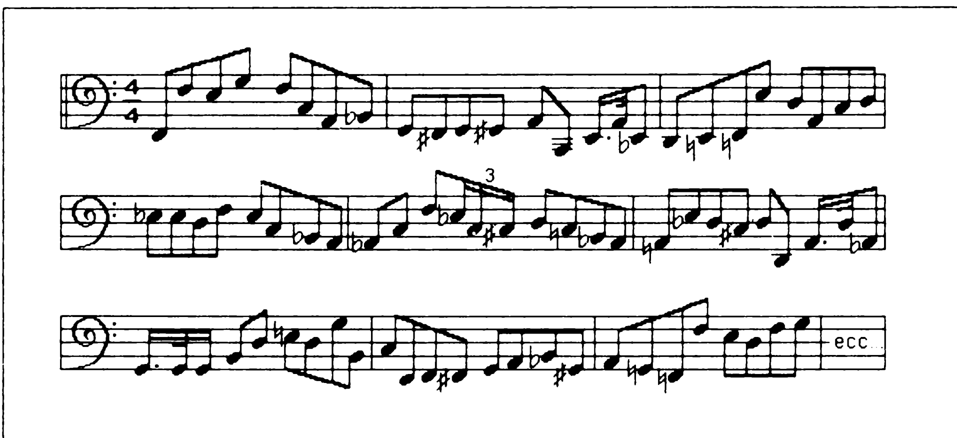



**Figura 10.1** Alcuni righi di uno spartito con la base per della musica jazz

minciare, definiamo la stringa in modo che possa essere suonata con un tempo arbitrario e senza interruzioni quando l'istruzione PLAY è eseguita come parte di un loop in un programma. A questo fine, inseriamo MB come primo comando nella stringa, ed assegnamo al comando T il valore di una variabile:

```
PLAY "MBT=TEMPO;"
```

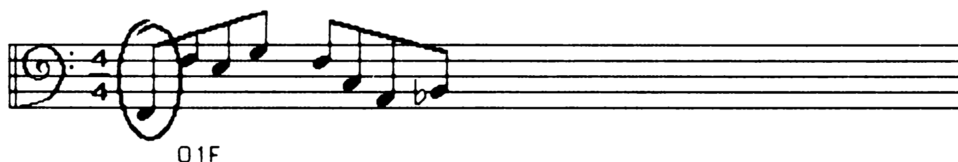
Dal momento che stiamo scrivendo un accompagnamento di basso, utilizziamo la notazione con il Legato:

```
PLAY "MBT=TEMPO;ML"
```

La maggioranza delle note che compaiono sono di un ottavo e perciò definiamo come durata di default il valore L8:

```
PLAY "MBT=TEMPO;MLL8"
```

Come vedremo, ogni nota che non sia un ottavo verrà seguita dal proprio specificatore di durata.
La prima nota è un Fa, due ottave sotto il Do centrale, cioè nell'ottava 1. Non ci sono in chiave né diesis né bemolle e perciò il Fa è naturale:




Aggiungendo questa prima nota la stringa diventa:

```
PLAY "MBT=TEMPO;MLL8O1F"
```

Al termine della prima battuta otteniamo, operando in questo stesso modo:

```
PLAY "MBT=TEMPO;MLL8O1FO2FEGFCO1AB-"
```

Osservate che dobbiamo passare dall'ottava 1 all'ottava 2 già alla seconda nota della battuta, un altro Fa naturale. Dobbiamo poi ritornare all'ottava 1 per la penultima nota, un La naturale; questa ottava è corretta anche per l'ultima nota, un Si bemolle.
Le prime sei note della seconda battuta sono semplici da trascrivere:

```
PLAY "MBT=TEMPO;MLL8O1FO2FEGFCO1AB-GF#GG#AO0A"
```

Il ritmo cambia, però, alla settima nota di questa battuta: ci troviamo di fronte ad un sedicesimo puntato, seguito da un trentaduesimo. Queste note saranno rappresentate così:



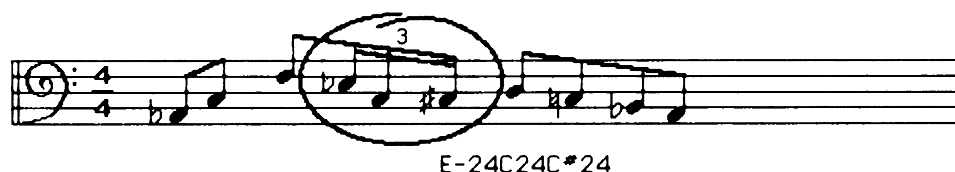

La stringa dell'istruzione PLAY ora appare così:

```
PLAY "MBT=TEMPO;MLL8O1FO2FEGFCO1AB-GF#GG#AOOA O1E16.
A32"
```

La trascrizione è lineare da qui fino alla metà della quinta battuta; fino a questo punto, la stringa si era così modificata:

```
PLAY "MBT=TEMPO;MLL8O1FO2FEGFCO1AB-GF#GG#AOOA O1E16.
A32E-DEFO2EDO1AO2CDMNE-MLE-DFECO1B-AA-O2CF"
```

Osservate che due consecutivi Mi bemolle all'inizio della quarta battuta sono stati resi diversamente: il primo è preceduto dal comando MN (Normale) ed immediatamente seguito da ML (Legato); questo fa sì che le due note siano suonate separatamente, altrimenti, con il Legato, verrebbero suonate come un singolo quarto.
Nel centro della quinta battuta dobbiamo codificare una tripletta: questi tre sedicesimi corrispondono a tre note della durata di 1/24 di battito. Possiamo trascriverlo in questo modo:




La stringa ora si presenta così:

```
PLAY "MBT=TEMPO;MLL8O1FO2FEGFCO1AB-GF#GG#AOOA O1E16.
A32E-DEFO2EDO1AO2CDMNE-MLE-DFECO1B-AA-O2CF E-24C24C#
24"
```

Trascrivendo il resto delle prime otto battute otteniamo:

```
PLAY "MBT=TEMPO;MLL8O1FO2FEGFCO1AB-GF#GG#AOOA
O1E16.A32E-DEFO2EDO1AO2CDMNE-MLE-DFECO1B-AA-O2CF
E-24C24C#24DCO1B-A-AO2E-DC#DO1DA16.O2D32O1A-
MNG16.MLG32ABO2DEDGO1BO2CO1EFF#GAB-G#"
```

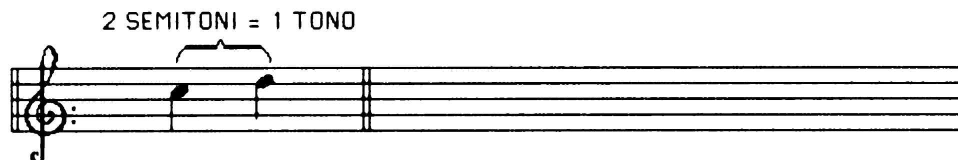
Un programma che debba eseguire questa stringa richiede innanzitutto che sia inizializzata la variabile TEMPO e poi l'esecuzione può essere ripetuta inserendo l'istruzione PLAY in un loop.

## 10.5 Come sviluppare un programma musicale

Cercheremo ora di creare un programma più sofisticato utilizzando l'istruzione PLAY: questo programma sarà in grado di eseguire una qualsiasi delle varie scale possibili (maggiore, minore, diminuita, e così via) in una qualsiasi delle 12 chiavi e con tempo a scelta.
La costruzione della scala di output è basata sugli intervalli tra le note della scala scelta. Un intervallo è la distanza tra due note consecutive; per esempio, l'intervallo tra un Do ed un Do diesis è di un semitono:




L'intervallo tra un Do e un Re, invece, è di due semitoni:
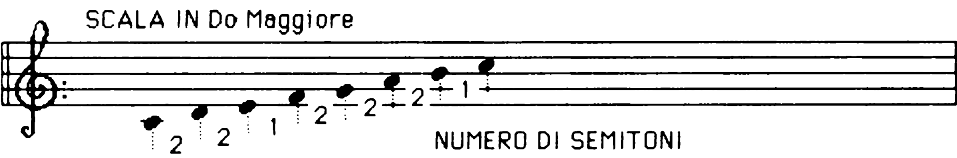



Un determinato tipo di scala — una scala maggiore, ad esempio — è definita da uno specifico insieme di intervalli, senza riferimento alcuno alla chiave in cui la scala è scritta. Ciò significa che possiamo costruire un dato tipo di scala in ogni chiave, semplicemente partendo dalla nota voluta e aggiungendo via via gli intervalli opportuni. Gli intervalli di una sca-

**Tabella 10.2** Intervalli di una scala Maggiore

| Intervallo | Numero di semitoni |
|---|---|
| Tra la prima e la seconda nota | 2 |
| Tra la seconda e la terza nota | 2 |
| Tra la terza e la quarta nota | 1 |
| Tra la quarta e la quinta nota | 2 |
| Tra la quinta e la sesta nota | 2 |
| Tra la sesta e la settima nota | 2 |
| Tra la settima e l'ottava nota | 1 |

la Maggiore sono riportati nella Tabella 10.2; se applicati alla chiave di Do, questi intervalli danno luogo alla scala di Do Maggiore, così come viene mostrato:




## ORGANIZZAZIONE DEL PROGRAMMA

Per creare questo programma, dobbiamo dapprima progettarne la struttura; la parte principale è formata da numerose chiamate a subroutine, come potete vedere:

```
      .
      .
      .
150 GOSUB 240 ' Visualizza il menu principale
160 GOSUB 580 ' Visualizza il secondo menu
170 GOSUB 780 ' Abilita l'intercettamento dei tasti
funzione
180 GOSUB 1180 ' Costruisce la scala
190 GOSUB 1340 ' Esegue la scala
      .
      .
      .
```

Ogni subroutine esegue un compito specifico: la prima, nella linea 240, visualizza il menu principale, richiedendo all'operatore il tipo di scala

che vuole eseguire e controlla l'input fornito; la routine successiva, nella linea 580, richiede all'operatore la chiave in cui suonare e di nuovo controlla l'input. Ai tasti funzione da F1 a F5 vengono assegnati compiti specifici nella subroutine che inizia alla linea 780; infine, viene costruita la stringa per l'istruzione PLAY, seguendo le richieste fatte, nella subroutine che parte dalla linea 1180. La stringa viene finalmente suonata dalla subroutine che inizia nella linea 1340.

## COME DEFINIRE L'INIZIO DELLA SCALA

Dopo che l'operatore ha selezionato il tipo di scala e la chiave in cui suonarla, la subroutine nella linea 580 determina la nota di partenza della scala con il seguente loop:

```
       .
       .
       .
630 RESTORE 730
640 FOR I=25 TO 36
650 READ TEST$
660  IF K$=TEST$ THEN GOOD=1:ROOT=I
670 NEXT I
       .
       .
       .
730 DATA "C","C+","D","E-","E","F","F+",
         "G","G+","A","B-","B"
       .
       .
       .
```

Questo loop confronta l'input corrente K$ con le scelte possibili elencate nella linea 730. Gli indici del loop, da 25 a 36, vengono confrontati con i numeri delle note mentre questi sono controllati. Quando si trova una corrispondenza, la variabile ROOT assume il valore dell'indice corrente del contatore di loop I: questa variabile può così essere usata direttamente come prima nota della scala che verrà poi costruita dal programma.

## COME COSTRUIRE LA SCALA

L'istruzione PLAY che rappresenta la scala effettiva viene costruita nella subroutine che inizia alla linea 1180, che usa questi due loop per costruire una versione discendente ed una ascendente della stessa scala:

```
        .
        .
        .
1220 FOR Z=1 TO STEPS
1230        NOTE(Z)=NOTE(Z-1) + INTERVAL(Z)
1240        SCALE$=SCALE$ + "n" + STR$(NOTE(Z))
1250 NEXT Z
1260 FOR Z=STEPS-1 TO 1 STEP -1
1270        NOTE((2*STEPS)-Z)=NOTE((2*STEPS)-Z-1)
            - INTERVAL(Z)
1280        SCALE$=SCALE$ + "n" + STR$(NOTE(Z))
1290 NEXT Z
        .
        .
        .
```

Notate che gli indici di questo loop dipendono dal valore della variabile STEPS, che viene inizializzata al momento della scelta del tipo di scala all'inizio del programma, poiché ogni particolare tipo di scala contiene un determinato numero di note. La scala Maggiore è composta da sette note, mentre la scala tonale ha solo sei note.
Osservate anche che l'array INTERVAL è riempito all'inizio del programma con i valori che indicano il numero di semitoni tra le note per la scala scelta.
L'istruzione PLAY per la scala ascendente viene costruita semplicemente aggiungendo gli appropriati intervalli, presi dall'array INTERVAL, al valore della nota precedente e poi convertendo questo numero nella rappresentazione in formato stringa. Osservate che alla nota iniziale della scala, NOTE(0), viene assegnato il valore della variabile ROOT, definita precedentemente dal programma.
In modo analogo, la stringa per l'istruzione PLAY che suona la scala discendente viene prodotta sottraendo gli opportuni intervalli al valore della nota precedente e poi convertendo il numero nella rappresentazione in formato stringa.

## IL PROGRAMMA COMPLETO

Il programma finale è proposto nella Figura 10.2: se volete capire più da vicino come il programma costruisca le scale, aggiungete le seguenti istruzioni PRINT per seguirlo mentre esegue i calcoli:

```
1205 PRINT "ROOT =" ROOT
1235 PRINT "NOTA #" Z " =" NOTE(Z)
1275 PRINT "NOTA #" Z " =" NOTE(Z)
```

Queste istruzioni visualizzano il numero della nota della scala non appena questo viene calcolato.

```
10 ' ************************************************
20 '
30 '    Introduzione all'uso delle scale musicali
40 '
50 '  Questo programma mostra l'uso dell'istruzione
60 '  PLAY con parametri variabili e sottostringhe.
70 '  Dovete eseguire questo programma in Advanced
80 '  BASIC.
90 '
100 '************************************************
110 '
120 DIM SCALA$(25),NOTE(20),INTERVAL(25)
130 '
140 '
150 GOSUB 240  'Visualizza il menu principale
160 GOSUB 580  'Visualizza il menu secondario
170 GOSUB 780  'Abilita l'intercettamento dei tasti
 funzione
180 GOSUB 1180 'Costruisce la scala
190 GOSUB 1340 'Esegue la scala
200 '
210 '
220 ' Visualizza il menu principale, richiede la scel
ta della scala e controlla l'input
230 '
240 CLS:KEY OFF
250 PRINT TAB(15)  "DIMOSTRAZIONE DELLA SCALA"
260 PRINT
270 PRINT
280 PRINT TAB(10)  "DECIDI QUALE SCALA VUOI SENTIRE"
290 PRINT :PRINT " 1. MAGGIORE"
300 PRINT " 2. ARMONICA MINORE"
310 PRINT " 3. DIMINUITA"
320 PRINT " 4. SEMI-DIMINUITA"
330 PRINT " 5. TONICA"
340 PRINT " 6. BLUES"
350 PRINT
360 PRINT
370 INPUT ;"   INSERIRE LA SCALA SCELTA (1-6) OPPURE
7 PER USCIRE - ",S
380 ON S GOTO  420, 430, 440, 450, 460, 470, 540
390 PRINT :PRINT "CATTIVA SCELTA, PROVA ANCORA!"
400 FOR X=0 TO 1000: NEXT
410 GOTO 220
```

(continua)

```
420 RESTORE 1400: STEPS=7: GOTO 480 ' "STEPS" e' il
numero di intervalli in ogni scala; la scala blues ad
esempio e'composta da sei note.
L'istruzione RESTORE punta alla istruzione DATA con
l'intervallo per ogni scala
430 RESTORE 1410: STEPS=7: GOTO 480
440 RESTORE 1420: STEPS=8: GOTO 480
450 RESTORE 1430: STEPS=7: GOTO 480
460 RESTORE 1440: STEPS=6: GOTO 480
470 RESTORE 1450: STEPS=6
480 FOR I=1 TO STEPS 'Riempie il vettore che contiene
gli intervalli della scala scelta.
490     READ INTERVAL(I)
500 NEXT I
510 RETURN
520 '
530 '
540 END ' **** TERMINA IL PROGRAMMA ****
550 '
560 '
570 ' Visualizza il menu con le chiavi e richiede
l'input
580 CLS
590 PRINT "INSERIRE LA CHIAVE IN CUI VOLETE LA SCALA
(O LA PRIMA NOTA)"
600 PRINT "  (C,C+,D,E-,E,F,F+,G,G+,A,B-,B) - "
610 INPUT ; K$
620 GOOD=0  'Flag per il controllo dell'input
630 RESTORE 730 ' Punta agli input validi
640 FOR I=25 TO 36
650   READ TEST$
660   IF K$=TEST$ THEN GOOD=1:ROOT=I ' Se viene trova
ta la corrispondenza per l'input, definisce il flag
di  controllo e ROOT uguale al numero della nota in
input
670 NEXT I
680 IF GOOD=0 THEN 700
690 RETURN
700 PRINT :PRINT   "CATTIVA SCELTA, PROVA ANCORA!!"
710 FOR X=0 TO 1000: NEXT X
720 GOTO 580
730 DATA "C","C+","D","E-","E","F","F+","G","G+","A",
"B-","B"
740 '
750 '
760 ' **** Abilita i tasti funzione ****
770 '
780 LOCATE 20,1
790 CLS:PRINT "KEY 1= RITARDANDO"
```

(continua)

```
800 PRINT "KEY 2= ACCELERANDO"
810 PRINT "KEY 3= MODULAZIONE SU MEZZO TONO"
820 PRINT "KEY 4= INSERIRE UNA NUOVA SCALA"
830 PRINT "KEY 5= FINE"
840 KEY (1) ON
850 KEY (2) ON
860 KEY (3) ON
870 KEY (4) ON
880 KEY (5) ON
888         LIST
890 ON KEY (1) GOSUB 980
900 ON KEY (2) GOSUB 1030
910 ON KEY (3) GOSUB 1080
920 ON KEY (4) GOSUB 1130
930 ON KEY (5) GOSUB 540
940 RETURN
950 '
960 ' **** Routine ritardando ****
970 '
980 TEMPO = TEMPO-20: IF TEMPO < 32 THEN TEMPO=32
990 RETURN
1000 '
1010 ' **** Routine accelerando ****
1020 '
1030 TEMPO = TEMPO+20: IF TEMPO > 255 THEN TEMPO=255:
 RETURN
1040 RETURN
1050 '
1060 ' **** Routine modulazione ****
1070 '
1080 ROOT = ROOT+1:GOSUB 1180
1090 RETURN
1100 '
1110 ' **** Routine per la nuova scala ****
1120 '
1130 GOSUB 220: GOSUB 580: GOSUB 1180: GOSUB 780
1140 RETURN
1150 '
1160 ' ***** Costruisce la scala *****
1170 '
1180 SCALA$=""
1190 TEMPO=80
1200 NOTE(0)=ROOT
1210 SCALA$="n" + STR$(NOTE(0)) 'Inizia la scala con
la nota ROOT
1220 FOR Z=1 TO STEPS
1230 NOTE(Z)=NOTE(Z-1) + INTERVAL(Z) 'Aggiunge a cias
cun tono della scala l'opportuno intervallo.
```

(continua)

```
1240 SCALA$=SCALA$ + "n" + STR$(NOTE(Z))
1250 NEXT Z
1260 FOR Z=STEPS-1 TO 1 STEP -1
1270 NOTE((2*STEPS)-Z)=NOTE((2*STEPS)-Z-1) - INTERVAL
 (Z) ' Costruisce la meta' discendente della scala.
1280 SCALA$=SCALA$ + "n" + STR$(NOTE(Z))
1290 NEXT Z
1300 RETURN
1310 '
1320 ' ***** ESEGUE LA SCALA *****
1330 '
1340 PLAY "MNMNL16T" + STR$(TEMPO) + SCALA$  'Esegue
la scala, preceduta dai parametri per il sottofondo,
in modo normale e con note di un sedicesimo.
1350 GOTO 1340
1360 '
1370 '
1380 ' ***** Intervalli per le scale *****
1390 '
1400 DATA 2,2,1,2,2,2,1 : ' Scala maggiore
1410 DATA 2,1,2,2,1,3,1 : ' Scala minore
1420 DATA 1,2,1,2,1,2,1,2 : ' Scala diminuita
1430 DATA 1,2,2,1,2,2,2 : ' Scala semidiminuita
1440 DATA 2,2,2,2,2,2 : ' Scala tonica
1450 DATA 3,2,1,1,3,2 : ' Scala blues
1460 '
```

**Figura 10.2** Programma di introduzione all'uso delle scale musicali

Capitolo

# Comunicazioni 11

Finora abbiamo preso in considerazione solo le comunicazioni tra il PC ed altri dispositivi come la tastiera ed i drive, effettuate per mezzo dei comandi del BASIC e del sistema operativo DOS. Questo capitolo vi mostra come scambiare informazioni tra il PC ed un grande numero di altri dispositivi, tra i quali altri PC, ed anche come usare schede di comunicazione seriali — come l'interfaccia per le comunicazioni asincrone — con programmi per particolari applicazioni che potete trovare in commercio o creare da voi.

## 11.1 Il processo della comunicazione

Per poter comunicare con dispositivi esterni dovete disporre di:

— Un'interfaccia IBM per comunicazioni asincrone (di serie sull'XT) o un'altra scheda seriale di comunicazione
— Un cavo di collegamento per la comunicazione tra il PC ed il dispositivo
— Un programma di supporto per la comunicazione

Il diagramma a blocchi di Figura 11.1 mostra le varie interconnessioni tra le parti appena nominate.
Il PC, naturalmente, controlla l'intero processo: l'interfaccia per comunicazioni asincrone (ACA) o un'altra scheda seriale convertono i dati provenienti dal PC nel formato corretto per l'invio ad altri dispositivi e viceversa; il collegamento è un cavo, o altro mezzo elettronico, che unisce

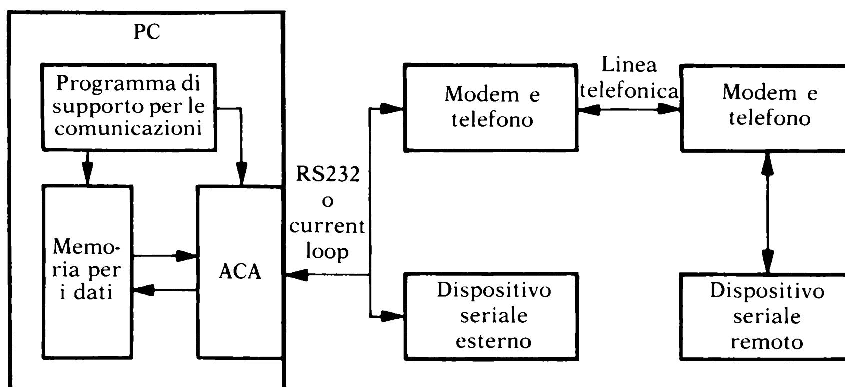



**Figura 11.1** Diagramma a blocchi delle comunicazioni

l'ACA con il dispositivo esterno. Quest'ultimo dev'essere in grado di ricevere dati dal PC, o trasmettere dati al PC o eseguire entrambe le operazioni. Infine, il programma di supporto controlla il trasferimento dei dati da comunicare tra la memoria del PC e l'ACA.

## INTERFACCIA PER COMUNICAZIONI ASINCRONE (ACA)

L'ACA viene fornito di serie sulla versione XT, mentre è opzionale per il PC. La scheda viene comunque installata nell'unità di sistema, agisce come un trasduttore, prendendo i dati provenienti dalla memoria ed inviandoli all'esterno nel formato specifico che il dispositivo esterno è in grado di comprendere. In modo analogo, l'ACA accoglie i dati provenienti dall'esterno e li traduce prima di immagazzinarli nella memoria del PC. Per garantire la compatibilità con i dispositivi esterni, l'ACA ha alcuni circuiti elettrici di interfaccia che si adattano agli standard di comunicazione seriale dell'RS232 e del current loop.
Diverse case costruttrici producono schede per comunicazioni seriali che possono essere usate al posto dell'ACA: controllate prima le specifiche di queste schede se avete delle esigenze particolari.

### Comunicazioni seriali asincrone

L'ACA comunica con il mondo esterno con un formato di comunicazione seriale ed asincrono: seriale significa che i dati sono inviati o ricevuti bit per bit attraverso una coppia di fili; asincrono significa che non c'è un tempo di riferimento comune tra chi invia i dati e chi li riceve, eliminan-

do così la necessità di un impulso di sincronizzazione comune tra i due dispositivi.
Questo metodo rende il collegamento tra il PC ed un altro dispositivo molto semplice: se i dati devono viaggiare in una sola direzione sono richiesti almeno due fili, mentre se la comunicazione deve passare nei due sensi il numero minimo di fili necessario è tre, anche se spesso ne vengono usati di più.
Comunicazione seriale asincrona si riferisce quindi ad un modo specifico di formattazione dei dati e non al collegamento fisico tra i dispositivi. L'ACA vi permette di scegliere tra due metodi standard che si uniformano alle normali specifiche elettriche e meccaniche: questi sono lo standard RS232 e lo standard current loop.
Un gruppo di microswitch, mostrato nella Figura 11.2, permette di scegliere lo standard da usare; normalmente, comunque, l'ACA viene venduta già predisposta con l'opzione RS232.

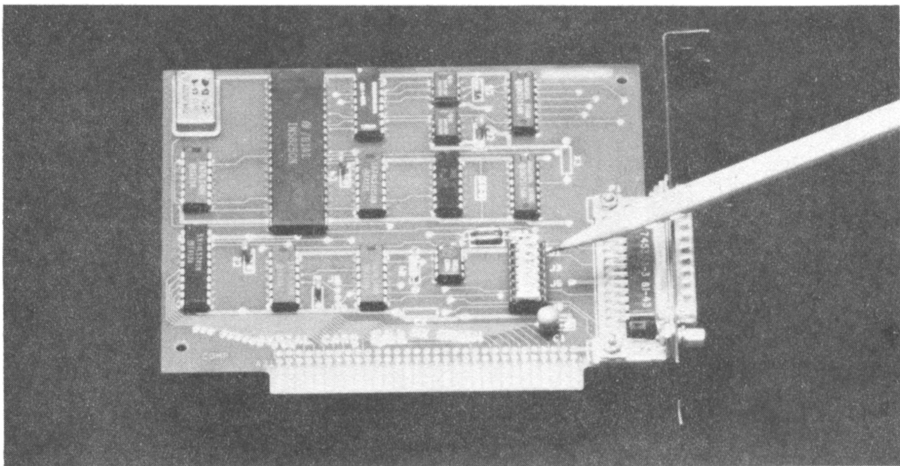


**Figura 11.2** Gli switch dell'ACA

Le varie interfacce per dispositivi seriali vengono pubblicizzate sotto vari nomi dalle diverse case costruttrici; vi forniamo una lista, seppur parziale, dei termini usati per far riferimento a questo tipo di interfaccia: seriale, seriale asincrona, RS232, RS232C current loop e EIA current loop.
Prestate attenzione al fatto che la maggior parte di queste interfacce, ma non tutte, si adatta alle richieste standard RS232 o current loop dell'ACA; se avete dubbi su di un particolare dispositivo, chiedete dettagliate spiegazioni al rivenditore.

**RS232**

L'interfaccia RS232 dell'ACA usa un connettore a 25 pin tipo D per collegare il PC con dispositivi seriali esterni. Il connettore è mostrato nella Figura 11.3.

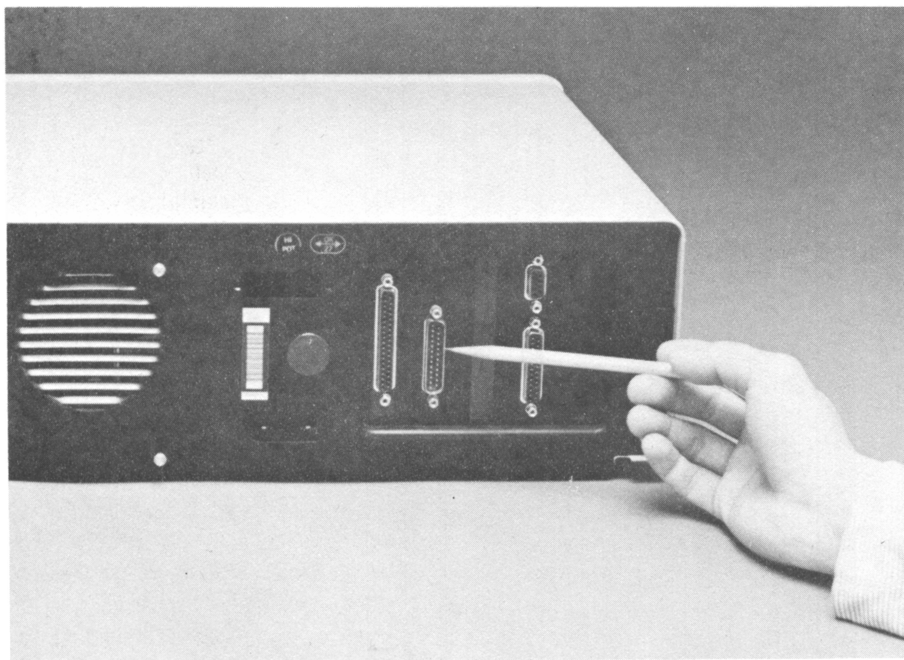
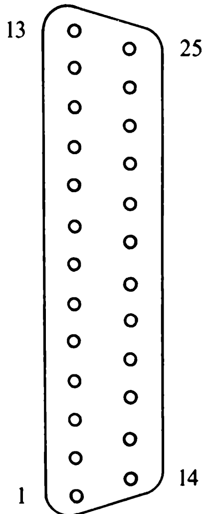


Figura 11.3 Il connettore D dell'ACA

Nella Figura 11.4 vengono definiti i nomi per i pin del connettore dell'ACA relativi ai segnali di controllo e ai dati per l'RS232. Solo alcuni dispositivi richiedono i collegamenti per i segnali di controllo. Per queste informazioni sarà bene che facciate riferimento al manuale operativo dei singoli dispositivi.
Osservate che viene usato un filo per il flusso di dati che vanno dal PC al dispositivo esterno (*Transmit Data*), mentre un altro viene usato per il flusso opposto, cioè dei dati che dal dispositivo esterno giungono al PC (*Receive Data*). Di solito, il cavo tra PC e dispositivo mette in collegamento gli stessi pin dei due connettori D, cioè il pin 1 ad un estremo sarà collegato con il pin 1 all'altro, e così via.
All'estremità collegata con il PC, l'ACA si aspetta i dati dal pin numero 3

Numerazione dei pin
del connettore D
dell'ACA




| | Numero del pin | Segnale |
|---|---|---|
| Linee dati | 2<br>3<br>7 | Transmit Data<br>Receive Data<br>Signal Ground |
| Linee di controllo | 4<br>5<br>6<br>8<br>20<br>22 | Request to Send<br>Clear to Send<br>Data Set Ready<br>Carrier Detect<br>Data Terminal Ready<br>Ring Indicate |

**Figura 11.4** Definizione dei segnali dell'RS232 per il connettore dell'ACA

del connettore D e li trasmette dal pin 2; a volte può accadere che un dispositivo invii dati dal pin 2 e li attenda in ingresso dal pin 3.
In questo caso, i cavi che connettono i due dispositivi devono invertire i due collegamenti di trasmissione-dati e ricezione-dati: il pin 2 all'estremità del PC deve essere collegato con il pin 3 all'altra estremità, mentre il pin 3 vicino al PC con il pin 2 dalla parte del dispositivo. Nella Figura 11.5 è mostrato un disegno di come devono essere disposti i fili in un cavo in cui trasmissione e ricezione vanno invertite; se doveste avere problemi con il vostro collegamento, potete provate ad adottare questa soluzione.

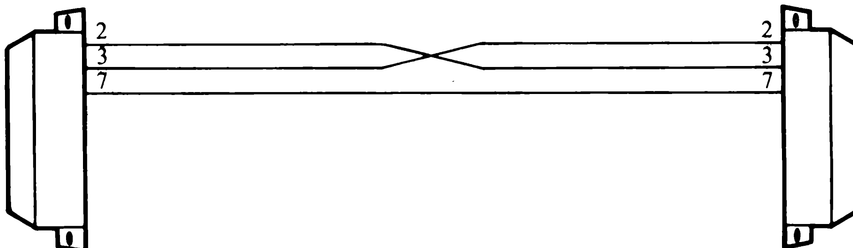
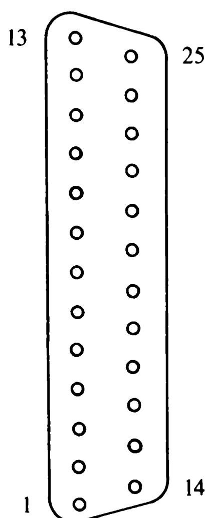



**Figura 11.5** Cavi di un'interfaccia RS232 in cui è invertita la linea di trasmissione con quella di ricezione

## Current loop

Il current loop richiede una coppia di fili per ogni direzione di flusso dei dati: queste linee di trasmissione vengono collegate al connettore di tipo D dell'ACA, come mostrato nella Figura 11.6. Il current loop viene solitamente impiegato nei vecchi modelli di telescrivente ed in alcune stampanti IBM.




| Numero del pin | Segnale |
|---|---|
| 9 | Transmit Return (+) |
| 11 | Transmit Data (−) |
| 18 | Receive Data (+) |
| 25 | Receive Return (−) |

**Figura 11.6** Definizione dei segnali sul connettore D dell'ACA per il current loop

Per il resto del capitolo ci riferiremo ad entrambi i tipi di dispositivi seriali asincroni, cioè RS232 e current loop, come a "dispositivi seriali".

## COLLEGAMENTI

Il modo più semplice per collegare il PC con un altro dispositivo è un cavo RS232, che può essere lungo fino a qualche decina di metri. Osservate che il connettore di tipo D ad un'estremità del cavo dev'essere "femmina" per potersi adattare all'ACA, mentre l'altra estremità deve portare un connettore "maschio" o "femmina" a seconda del dispositivo da collegare.

Il cavo adatto viene usualmente venduto insieme al dispositivo seriale esterno da collegare al PC; in caso contrario ne potete trovare facilmente uno da un rivenditore.
Tra i dispositivi che possono utilizzare un collegamento seriale troviamo:

— Stampanti
— Terminali video
— Plotter
— Equipaggiamenti per acquisizione dati
— Altri personal computer
— Minicomputer
— Grandi elaboratori (Mainframe)

La maggior parte dei dispositivi può utilizzare sia lo standard RS232 che il current loop, ma vi conviene sempre controllare la documentazione originale quando preparate il cavo per connetterne uno al PC.

### Modem

Il PC può anche comunicare con dispositivi remoti per mezzo di normali linee telefoniche; in questo caso, però, dovete collegare sia il PC che il dispositivo remoto ad un modem. Il modem traduce i segnali in uscita o in entrata dall'ACA in suoni che possono essere trasmessi attraverso le linee telefoniche: una configurazione tipica è quella mostrata nella Figura 11.1. La Figura 11.7, invece, mostra due tipi di modem: acustico e a collegamento diretto.
Il modem ad accoppiatore acustico ha un sostegno nel quale viene posta una cornetta telefonica; il modem a collegamento diretto ha una spina che si inserisce direttamente in una presa telefonica modulare. Entrambi

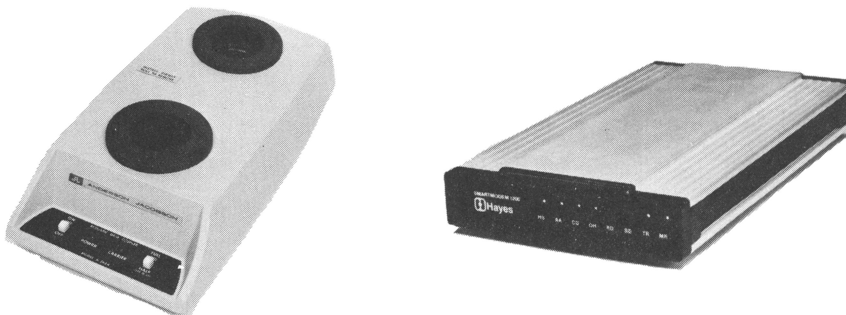



**Figura 11.7** Un modem acustico ed uno a collegamento diretto

i tipi di modem dispongono di un connettore D che può ricevere dati dal PC o da un dispositivo seriale.
I modem possono essere suddivisi in tre categorie: i modem di richiesta, i modem di risposta e quelli ambivalenti. Un modem di richiesta viene utilizzato per iniziare il collegamento telefonico, mentre un modem di risposta è predisposto per rispondere al segnale del modem di richiesta. Per le applicazioni in cui serva chiamare un computer od un dispositivo remoto, è necessario un modem del primo tipo, mentre quando sia il computer remoto a cui voi dovete rispondere, viene naturalmente utilizzato un modem di risposta. Molti modem comprendono ambedue le funzioni, di richiesta e di risposta.
Per usare un modem allo scopo di chiamare un computer remoto, in genere si opera in questo modo: viene avviato il programma di comunicazione del PC; se si usa un modem a collegamento diretto, il telefono dev'essere collegato al modem prima di comporre il numero. Dopo aver composto il numero opportuno, il computer remoto risponde alla telefonata con un suono e a questo punto, se state usando un accoppiatore acustico, inserite la cornetta del telefono nell'apposito vano e iniziate il collegamento.
Naturalmente, l'esatta procedura da seguire dipende in larga misura dal software di comunicazione, dal tipo di modem e dal dispositivo seriale.

## PROGRAMMA DI SUPPORTO PER LE COMUNICAZIONI

Il programma di supporto delle comunicazioni controlla il flusso dei dati tra la memoria del PC e l'ACA; può anche trasferire i dati sul video o immagazzinarli su disco per un uso successivo.
Sono diverse le soluzioni possibili per il software per le comunicazioni:

— Programmi per comunicazioni prodotti o no dall'IBM
— Un programma BASIC scritto da voi
— Programmi in Assembler scritti da voi

Questo capitolo focalizza l'attenzione sull'uso dei programmi che si possono acquistare già pronti e sulla migliore maniera di scrivere i propri programmi per comunicazioni in BASIC. La possibilità di scrivere programmi di comunicazione in Assembler richiede, invece, una conoscenza approfondita della macchina e dell'ACA.

### Programmi di comunicazione IBM

L'IBM fornisce due programmi di supporto per comunicazioni seriali: uno è il programma COMM.BAS; l'altro è il programma di supporto per

comunicazioni asincrone (ACS), che viene venduto separatamente insieme ad un manuale di riferimento.
I due programmi, COMM e ACS, sono simili, ma vengono impiegati diversamente: la differenza principale sta nel fatto che il programma ACS vi permette di trasferire automaticamente file di dati, mentre il programma COMM fa sì che il PC si comporti come un semplice terminale, da cui i dati possono essere trasferiti solo byte per byte.
Il programma COMM permette queste applicazioni:

— Accedere a un database, come The Source o CompuServe
— Comunicare semplicemente con un altro PC
— Scambiare informazioni con un computer IBM Serie/1 che usi uno di questi due sistemi operativi: Real Time Programing System V5.1 o Event Driven Executive V3.0
— Comunicare semplicemente con un altro dispositivo seriale

Il programma ACS, invece, consente queste applicazioni:

— Comunicare con un computer IBM che stia lavorando sotto VM/370 o MVS TSO
— Realizzare comunicazioni sofisticate con un altro PC o con un altro dispositivo seriale

In seguito esamineremo anche un esempio di applicazione di questi programmi: il COMM per accedere a dati on-line e l'ACS per il trasferimento di file tra due PC.

## 11.2 Parametri per le comunicazioni seriali

Numerosi parametri controllano la formattazione dei dati per le comunicazioni seriali: sarà perciò utile familiarizzare con questi, dal momento che sia il programma di supporto per la comunicazione che il dispositivo esterno richiedono che i valori per questi parametri vengano indicati prima di attivare la comunicazione.
La predisposizione dei parametri dipende dal programma e dal dispositivo che state usando. Dalla parte del PC, un dato parametro può essere definito automaticamente dal programma per la comunicazione, oppure il programma stesso può richiedervi di inserire quell'informazione; nel modem o in un dispositivo esterno il parametro può essere definito già in fase di costruzione o regolato dall'utente per mezzo di più interruttori. I manuali del programma o del dispositivo in questione dovrebbero indicarvi i valori adatti ed il modo di definirli per questi parametri.
Molti dei problemi che insorgono nella comunicazione seriale sono causa-

ti da discrepanze tra i valori dei parametri del PC e quelli del dispositivo collegato: se vi accade di imbattervi in questi problemi, ricontrollate attentamente tutti i valori.
I parametri per le comunicazioni seriali che incontrerete più facilmente sono: il baud rate, il numero di bit di dati per parola, la parità ed il numero di bit di stop.

## BAUD RATE

Con il vocabolo *baud rate* si indica la velocità alla quale i dati vengono trasmessi; il parametro, detto anche *bit per secondo, bit rate, data rate,* assume normalmente uno di questi valori: 300, 600, 1200, 2400, 4800, 9600 o 19200.

## BIT DI DATI

Il numero di *bit di dati* si riferisce al numero di bit che costituiscono una singola word di dati o parola; nella comunicazione seriale, infatti, ogni word viene trasmessa come una sequenza di bit, come mostrato nella Figura 11.8. Se voi, per esempio, specificate che la lunghezza di una word è

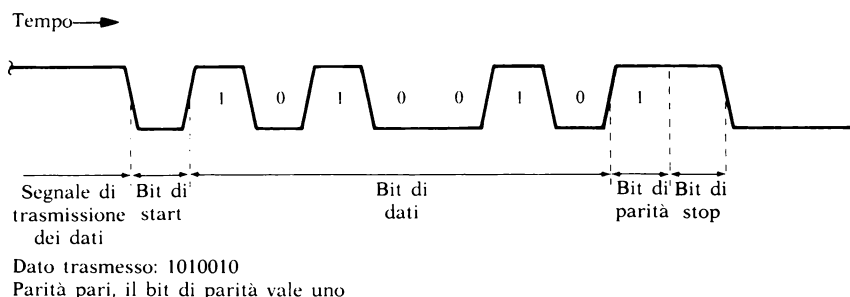



**Figura 11.8** La struttura di una word con sette bit di dati, parità pari, più un bit di stop

di 7, ogni singola word inviata sulla linea di comunicazione risulta composta da sette bit, uno dietro l'altro. Questa word può inoltre essere immediatamente seguita o preceduta da altri bit (bit di parità, o bit di start, oppure bit di stop) usati per il controllo della comunicazione. Il numero di bit di stop è variabile, mentre invece è sempre uno solo il bit di start.

## PARITÀ

La parità è uno dei metodi utilizzati per rivelare errori di trasmissione nella comunicazione dei dati. Il bit di parità viene aggiunto alla fine di ogni word ed il suo valore è funzione del resto della word stessa. Sono molti i modi in cui può essere calcolato il valore del bit di parità. *Parità pari* significa che il bit di parità deve avere un valore tale che la somma di tutti i bit della parola, compreso lo stesso bit di parità, sia un numero pari. Supponiamo di avere una word di sette bit di dati specificata con parità pari. Se la word è:

1010010

il bit di parità dev'essere 1, poiché questo rende pari il numero degli uno, che risulta essere così:

10100101

Se il dato originale fosse invece:

0001010

il bit di parità dovrebbe essere 0, per rendere pari la somma di tutti i bit. La word completa si presenta in questo modo:

00010100

La *parità dispari* è l'inverso: il bit di parità deve assumere un valore tale per cui la somma di tutti i bit in una word completa sia un numero dispari. La *parità di segno* indica che il bit di parità vale sempre 1, la *parità di spazio* che vale sempre 0, mentre *nessuna parità* significa che non viene aggiunto alcun bit di parità al termine della word.

## BIT DI STOP

Nelle comunicazioni seriali asincrone vengono sempre aggiunti, al termine della parola, uno o due bit di stop, che indicano al ricevente dove si conclude la word. Nella Figura 11.8 viene schematizzata una parola seriale di sette bit con parità pari ed un bit di stop.

### COMUNICAZIONI SERIALI FULL-DUPLEX E HALF-DUPLEX

Un'ulteriore caratteristica di un collegamento è se la connessione sia *full-duplex* o *half-duplex*. Spesso questo è uno dei parametri che vengono predisposti all'interno del programma per le comunicazioni, ma in alcuni casi dovete disporre opportunamente gli interruttori dei modem.
*Full* e *half* si riferiscono al numero di canali che compongono il collegamento seriale: full indica che ne sono presenti due, mentre half che c'è n'è uno solo. Con un half-duplex può "parlare" un solo dispositivo per volta, mentre con una comunicazione full-duplex, le informazioni possono fluire in entrambe le direzioni contemporaneamente. La maggior parte delle comunicazioni seriali con il PC avvengono per mezzo di collegamenti full-duplex.

## 11.3 Come accedere a banche dati on-line

Affrontiamo ora il problema di come collegare il PC ad una banca dati on-line utilizzando il programma COMM.BAS; in questo esempio, analizzeremo l'accesso a The Source, una delle maggiori banche dati USA.

### I COMPONENTI

Elenchiamo innanzitutto tutti i componenti che vi serviranno per realizzare l'accesso alle informazioni on-line:

— Un PC con almeno un drive ed un adattatore per comunicazioni asincrone
— Un modem (full-duplex; acustico o a collegamento diretto)
— Un cavo RS232 tra il PC ed il modem
— Un telefono (o una linea di trasmissione dati)
— Un conto aperto presso the Source
— Una copia del disco DOS

### IL PROCEDIMENTO

Il primo passo è di collegare i cavi dal PC al modem e, in caso di modem a collegamento diretto, anche dal modem alla linea TD; un'estremità del cavo RS232 va inserita nel connettore D a 25 pin dell'ACA che si trova sul retro del PC; l'altra invece nel connettore D che si trova sul modem. Se quest'ultimo è a collegamento diretto, sarà già collegato alla linea.
Avviate poi il PC dopo aver inserito nel drive A: il disco DOS; caricate ed

eseguite il programma COMM battendo, in risposta al prompt A>, le seguenti parole:

```
BASIC COMM
```

All'avvio del programma COMM verrà visualizzato un menu di comandi. Per una descrizione generale del programma, premete il tasto 1, mentre per comunicare con The Source premete 5; vi verrà richiesto il numero di telefono per il collegamento. Se possedete un modem acustico, attendete il suono ad alta frequenza e poi ponete il ricevitore nella sede apposita sul modem; se invece il modem è a collegamento diretto, dovete convertire il collegamento, di solito con un interruttore, da "fonia" a "dati". Dopo aver premuto ripetutamente il tasto ENTER, se il collegamento è stato stabilito correttamente, vi sarà richiesto di inserire un numero che identifica il tipo di terminale o di computer che state utilizzando: per il momento premete semplicemente il tasto ENTER. Quando appare sullo schermo il simbolo @, battete C xxxx, ove xxxx è il numero del sistema con cui volete collegarvi. In seguito a questa richiesta, The Source vi chiederà il vostro numero di riconoscimento (ID) e la password, dopo di che risulterete collegati a tutti gli effetti con The Source. A questo punto, seguendo semplicemente tutte le istruzioni fornite dal menu di The Source, potrete accedere a tutte le informazioni contenute in quel database.

## 11.4 Come trasferire file tra due PC

Come altro esempio di comunicazione seriale consideriamo il metodo per trasferire file tra due PC utilizzando il programma di supporto per l'interfaccia asincrona (ACS).
Il programma ACS trasferisce i file solo in formato testo, cioè ASCII, e solo se contengono linee lunghe non più di 254 caratteri. Potete perciò inviare e ricevere anche file di programmi BASIC, purché salvati in formato ASCII.

### I COMPONENTI

I componenti necessari per trasferire file tra due PC sono:

— Due PC con drive e ACA
— Un cavo per dati RS232 in cui i fili che sono collegati ad un'estremità ai pin 2 e 3 all'altra risultino invertiti (come nella Figura 11.5) e connettori "femmina" ad entrambe le estremità

oppure

— Una coppia di modem con i relativi cavi RS232 e la linea telefonica
— Due copie del programma ACS

## IL PROCEDIMENTO

Il primo passo è quello di collegare i due PC o per mezzo del modem o direttamente con il cavo RS232; successivamente avviare il programma ACS su entrambi i computer inserendo la copia del programma nel drive A: ed eseguendo una reinizializzazione del sistema per caricare il programma. (Osservate che, prima di poter utilizzare il programma ACS, dovete copiare il DOS sul disco contenente il programma, come descritto nel manuale ACS).
Dopo l'avvio, il programma ACS vi richiede di scegliere la larghezza dello schermo e, dopo la vostra risposta, mostra il menu principale. Per trasferire i file tra i PC, dovete scegliere l'opzione di comunicazione tra Personal Computer.
Vi viene poi fornita l'opportunità di modificare il baud rate dal valore di default di 300: il massimo valore raggiungibile dipende dal modem e dalla linea telefonica che state utilizzando.
Il passo successivo riguarda il modem: entrambi devono operare in modo full-duplex e naturalmente essere collegati ad un PC.
Entrambi gli operatori devono scegliere l'opzione di attivazione del collegamento. Solo al momento in cui sarà stabilito il collegamento sparirà dallo schermo il messaggio "Computer Connection NOT Established"; a questo punto uno dei due operatori chiama l'altro. Non appena il collegamento è raggiunto, i ricevitori vanno inseriti negli alloggiamenti dei modem acustici, oppure i modem a collegamento diretto devono essere passati dal modo "fonia" al modo "dati". Il PC dovrebbe rispondere a queste operazioni con il messaggio "Line Connected".
Per trasferire i file, l'operatore deve premere il tasto funzione F2, che fa visualizzare il menu di selezione delle funzioni. Uno dei due operatori deve scegliere l'opzione di trasmissione mentre l'altro quella di ricezione. Il programma ACS allora chiede il nome dei file, e dopo poco tempo visualizza un messaggio che indica che il trasferimento è iniziato. Questi file vengono inviati una linea alla volta, ed il numero della linea in trasmissione viene visualizzato sullo schermo.
Al termine del trasferimento, il messaggio "Transmission Completed" appare sullo schermo: il ricevente può interrompere la trasmissione premendo il tasto funzione F1 ed in questo caso il PC in trasmissione visualizza il messaggio "Transmission Ended at Request of Receiver" (trasmissione terminata su richiesta del ricevente).

Se si verifica una interruzione del trasferimento, ad esempio per la "caduta" della linea telefonica, potete di solito ritornare al menu principale premendo il tasto funzione F1 e da qui ricominciare il procedimento.
Il manuale d'uso esamina in maggiore dettaglio il processo di trasferimento di file, oltre che le altre funzioni del programma ACS.

## 11.5 Come scrivere i programmi di comunicazione in BASIC

Inviare dati all'esterno o riceverli utilizzando il BASIC è un processo molto simile a quello che permette di accedere a file su disco. Dovete inizializzare un file di comunicazione e poi accedere ad un buffer di comunicazione per l'input e l'output dei dati. Un programma BASIC può anche determinare lo stato del collegamento con le funzioni del buffer del file, proprio come è stato fatto nel Capitolo 7 con i file su disco. La Figura 11.9 mostra un diagramma dell'intero procedimento.

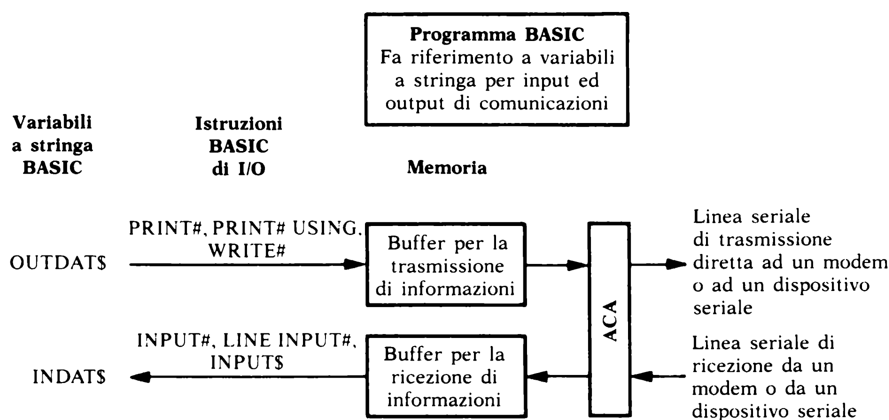



**Figura 11.9** Un processo di comunicazione con il BASIC

Per quanto riguarda il programma BASIC, l'operazione di accesso al buffer è un'estensione del processo di comunicazione. Come potete vedere nella Figura 11.9, tutti i trasferimenti tra il buffer e l'ACA e le operazioni di trasmissione e ricezione tra ACA e la linea seriale di comunicazione non sono controllate dal programma BASIC.
Esaminiamo ora la costruzione di un programma BASIC per comunicazioni; lo svilupperemo sezione per sezione, usando alcune proprietà del

BASIC per le comunicazioni. Il listato completo del programma è riportato alla fine del capitolo.
Osservate che vengono utilizzati solo alcuni strumenti di comunicazione del BASIC; per le altre istruzioni o funzioni potete far riferimento al compendio nell'Appendice A.

## COME PREDISPORRE IL BASIC PER LE COMUNICAZIONI

Quando avviate il BASIC dovete fare attenzione a due opzioni, molto importanti per quanto riguarda le applicazioni nel campo della comunicazione: sono le opzioni /S: e /C:, che si inseriscono in questo modo nella sintassi del comando BASIC:

> BASIC[A] ["*specfile*"] [<*stdin*] [[>]>*stdout*] [/F:*nfile*]
> [/S:*dimbuf*] [/C:*combuf*] [/M:[*max spaziolavoro*]
> [,*max dimbloc*]] [/D]

Il parametro /S:, come abbiamo visto nel Capitolo 7, si riferisce alla quantità di spazio riservato per i buffer dei file: in relazione ai file di comunicazione, il parametro è significativo solo nell'uso dei comandi GET e PUT con il buffer di comunicazione. (GET e PUT trasferiscono dati tra un buffer di comunicazione ed il buffer di un file ad accesso diretto). Il valore di questo parametro può essere al massimo 32767, mentre il valore di default è 128.
Il parametro /C: riserva lo spazio per i buffer dei file di comunicazione, spazio che può essere al massimo di 32767, mentre per default è di 256 byte per il buffer in ricezione e 128 per il buffer in trasmissione. Per linee ad alta velocità (per esempio, un baud rate maggiore di 1200), vi consigliamo di adottare un valore minimo di 1024 per questo parametro.

## COME UTILIZZARE I FILE BASIC PER LE COMUNICAZIONI

Il processo BASIC di comunicazione inizia con l'istruzione OPEN "COM, che definisce un buffer per la comunicazione e predispone i parametri di comunicazione. Non appena i dati sono disponibili, potete inviarli in output per mezzo delle istruzioni PRINT#, PRINT# USING, o WRITE#, proprio come per i file sequenziali, oppure con l'istruzione PUT che trasferisce un predeterminato numero di byte per volta.
Un programma BASIC può scoprire in due modi se i dati in input provenienti da un dispositivo esterno sono disponibili. Innanzitutto può controllare il buffer in input con la funzione EOF, che indica che sia stato o meno trovato qualche carattere in attesa di essere letto. Il secondo meto-

do prevede un'interruzione del programma provocata dall'interfaccia di comunicazione ogni volta che un dato sia disponibile e ciò viene fatto con l'abilitazione della routine di intercettamento indicata dall'istruzione ON COM.
I dati contenuti nel buffer di input vengono letti in modo sequenziale con le istruzioni INPUT#, INPUT$ o LINE INPUT#; per input di un numero determinato di caratteri utilizzate invece l'istruzione GET.
Il programma che tra breve verrà sviluppato si focalizza sull'uso dei comandi di accesso ai file sequenziali sia per l'input che per l'output di dati di comunicazione.

## UN ESEMPIO DI PROGRAMMA DI COMUNICAZIONE

Sviluppiamo ora un programma BASIC per comunicazioni, un programma che dimostri come comunicare a 1200 baud con uno strumento esterno collegato al PC con un cavo RS232. Viene considerata un'operazione con full-duplex, poiché sia il PC che lo strumento controllano una linea di trasmissione ed una di ricezione. Osservate che entrambe le linee devono avere i collegamenti dei pin 2 e 3 invertiti, cioè il pin 2 ad un capo del cavo dev'essere collegato all'altro capo al pin 3 e viceversa.
Questo programma di comunicazione dispone di quattro comandi: i primi tre inviano un codice ASCII di due caratteri allo strumento, in modo da accenderlo, spegnerlo o farlo rimanere in attesa di un comando sotto forma di parametro numerico. Il quarto comando serve sia per spegnere lo strumento che per terminare il programma. Dopo aver inviato il parametro di "attesa", il PC manda una sequenza di dati numerici, seguita da uno speciale carattere End-of-data (fine dei dati) che lo strumento sia in grado di riconoscere.
Lo strumento, inoltre, invia dati al PC a intervalli non predeterminati; i dati devono essere memorizzati su disco per poter essere elaborati in un secondo tempo. Lo strumento può anche inviare un codice d'errore al PC; se ciò avviene, il PC deve fermare lo strumento e ricominciare.
Ora esaminiamo il programma passo per passo.

### Inizializzazione

Innanzitutto dobbiamo predisporre una linea a 1200 baud, con una word di 8 bit (necessaria per dati di tipo numerico), nessuna parità ed un bit di stop. Questi parametri rispecchiano le esigenze dell'interfaccia di comunicazione dello strumento.
I parametri dell'istruzione OPEN "COM definiscono i valori per l'interfaccia per comunicazioni asincrone: potete specificare a quale interfaccia

del PC far riferimento, così come il baud rate, la parità, la lunghezza della word, il numero di bit di stop ed il numero del file di comunicazione. Ecco l'istruzione per inizializzare l'ACA con le nostre richieste:

```
20 OPEN "COM1:1200,N,8" AS #3
```

Questa forma dell'istruzione OPEN "COM fa riferimento all'interfaccia numero 1 e l'assegna al file numero 3. La N indica che non è specificata la parità, mentre 8 sta per 8 bit di dati. Un valore di default di 1 definisce il numero di bit di stop.
Per abilitare l'intercettamento di eventi, usiamo l'istruzione:

```
50 ON COM(1) GOSUB 480
```

Questa forma di ON COM significa che, ogniqualvolta venga rilevato un carattere in input nell'interfaccia numero 1, la routine alla linea 480 dev'essere eseguita. L'intercettamento di eventi di comunicazione viene automaticamente disabilitato all'inizio della routine in modo da evitare intercettamenti recursivi. L'intercettamento viene altrettanto automaticamente riabilitato all'esecuzione dell'istruzione RETURN al termine della routine.
Per aprire un file di dati sequenziale in cui memorizzare i dati provenienti dallo strumento, utilizziamo l'istruzione:

```
30 OPEN "ANALIZZA.DAT" FOR OUTPUT AS #2
```

**Come stampare il menu ed i comandi**

La prossima mossa è la presentazione all'operatore del menu dei comandi con una descrizione del significato e dell'uso di ogni comando. Il programma poi controlla la validità del comando inviato e salta all'opportuna routine di comando. Tutto ciò è realizzato dalle seguenti righe:

```
60 GOSUB 390 ' Visualizza il menu dei comandi
70 INPUT C: ON C GOTO 110,150,230,190 ' Scelta dell '
input, salta alla
75 ' routine dei comandi
80 PRINT "CATTIVA SCELTA, PROVA ANCORA" ' Messaggio
d'errore
90 FOR Q=0 TO 500:NEXT
100 GOTO 60
        .
        .
390 CLS
400 PRINT "INSERIRE LA SCELTA DEL COMANDO (1-4) - "
410 PRINT
420 PRINT "1. INIZIALIZZA LO STRUMENTO"
```

```
430 PRINT "2. SPEGNI LO STRUMENTO"
440 PRINT "3. INVIA DATI DI COMANDO"
450 PRINT "4. SPEGNI E TERMINA"
460 PRINT
470 RETURN
```

L'istruzione ON GOTO nella linea 70 provoca un salto che dipende dal valore di C : se questo non è compreso tra 1 e 4, viene eseguita la linea 80 che visualizza un messaggio d'errore.

**Routine di comandi**

Sono quattro le routine di comandi in questo programma: le prime due, INIZIALIZZA LO STRUMENTO e SPEGNI LO STRUMENTO, inviano semplicemente all'esterno sequenze di comandi di due caratteri che lo strumento è in grado di interpretare (A0 ed A1, rispettivamente). La routine di inizializzazione dello strumento, per esempio, è fatta così:

```
110 PRINT #3,"A0"
120 PRINT "STRUMENTO INIZIALIZZATO"
130 FOR Q=0 TO 500 : NEXT
140 GOTO 60
```

L'istruzione PRINT invia la stringa "A0" all'ACA, che a sua volta la invia allo strumento.
Il comando SPEGNI E TERMINA è identico al comando SPEGNI, eccetto per il fatto che include anche la linea:

```
220 CLOSE : END
```

che chiude sia il file di comunicazione che il file di dati sul disco e fa terminare l'esecuzione del programma.
Il comando INVIA DATI DI COMANDO è più complesso: la routine richiede all'operatore una sequenza di comandi, li visualizza come controllo, converte i dati di tipo stringa in dati numerici, invia il codice di inizio-dati (A2) allo strumento, seguito dai comandi ed infine termina inviando il codice fine-dati (A3). Ecco la routine di invio dei dati di comando:

```
230 CLS
240 LINE INPUT "INSERIRE DATI DI DUE CIFRE; ES. 39,00
,32 <ENTER> - ";COMDAT$
250 PRINT #3,"A2" ' "A2" e' il comando di attesa
260 PRINT " DATI STRUMENTALI - "
270 FOR Q=0 TO 500 :NEXT
280 Y=0
```

```
290 FOR X=1 TO (LEN(COMDAT$)+1)/3
300 Y=Y+1
310 Z$=MID$(COMDAT$,Y,2)
320 PRINT VAL(Z$)
330 PRINT #3, VAL(Z$)
340 Y=Y+2
350 NEXT
360 PRINT #3, "A3"
370 FOR Q=0 TO 900 : NEXT
380 GOTO 60
```

Osservate come la risposta, di due lettere, sia estratta dalla stringa in input e convertita nella rappresentazione numerica equivalente; osservate anche che viene aggiunto 1 all'intervallo del calcolo dell'istruzione FOR nella linea 290, che tiene conto della virgola mancante al termine della stringa in input.

### Come manipolare l'input proveniente dallo strumento

Nella linea 50 viene abilitato l'intercettamento della comunicazione per eseguire la routine alla linea 480. Questa routine ha diversi compiti: sospende l'output dello strumento se il buffer di input è pieno, legge i dati in input, controlla la presenza di un eventuale carattere-codice d'errore in arrivo dallo strumento, memorizza i dati nel file su disco e fa ripartire l'output dello strumento se l'aveva interrotto.

```
480 IF LOC(3)>128 THEN SOSPENDI=1:PRINT #3,XOFF$
490 A$=INPUT$(LOC(3),#3)
500 FOR I=1 TO LEN(A$)
510 IF MID$(A$,I,1)=CHR$(27) THEN 550
515 NEXT
520 PRINT #2,A$: IF EOF(3) THEN 530 ELSE 480
530 IF SOSPENDI=1 THEN SOSPENDI=0:PRINT #3,XON$
540 RETURN
550 PRINT "ERRORE DELLO STRUMENTO"
560 GOTO 150
```

La funzione LOC restituisce il numero di caratteri presenti nel buffer di input: se questo è troppo pieno, l'input dev'essere sospeso (con il carattere XOFF) in modo da non perdere alcun dato. Più avanti, nella linea 530, un flag (SOSPENDI) viene controllato per vedere se l'input era stato sospeso ed in tal caso uno speciale carattere (XON) viene inviato per far riprendere l'output dello strumento: questa tecnica di controllo dell'output di un dispositivo da parte di un altro viene detta *handshaking*.
Osservate che il programma deve assegnare valori alle variabili di tipo

stringa XOFF$ e XON$ prima di usarle. In questo esempio, sono stati usati i codici dei caratteri di controllo standard ASCII (19 e 17 rispettivamente), ma non tutte le interfacce li riconoscono.
Il nostro esempio prevede che lo strumento invii un carattere ESC (Escape, codice ASCII 27) se rileva un errore e deve essere fermato. La stringa A$ viene utilizzata per il carattere ESC e, se trovata, provoca l'esecuzione della linea 550.
Nella linea 520, viene usata la funzione EOF per controllare se qualche carattere è stato ricevuto mentre la stringa corrente A$ è stata elaborata. Questa funzione restituisce un valore −1 (vero) quando non ci sono caratteri in attesa e 0 (falso) se uno o più caratteri si trovano nel buffer di input. In questo modo, EOF viene qui usata per determinare se altri caratteri debbono essere letti dal buffer di comunicazione. Se non ci sono più caratteri il programma torna a visualizzare il menu dei comandi.

## Come usare l'istruzione INPUT$

Osservate che per l'input viene utilizzata in questo programma l'istruzione INPUT$; per tutti gli scopi di un programma di comunicazione è spesso più indicata delle istruzioni INPUT# e LINE INPUT#, poiché queste ultime filtrano i dati in arrivo in quanto sono in grado di riconoscere alcuni caratteri come delimitatori. Spesso, invece, tutti i caratteri che compongono un messaggio di comunicazione hanno un particolare significato per il programma che li riceve e perciò ogni carattere dev'essere mantenuto tra i dati in input. È questo il motivo per cui l'istruzione INPUT$ è utilizzata qui, al fine di ottenere esattamente quello che si trova nel buffer di input, fino ad un massimo di 255 caratteri.

## Il programma completo

Il programma di comunicazione che abbiamo descritto passo-passo, è presentato nella Figura 11.10.

```
1 'CONTROLLO DI UNO STRUMENTO E MEMORIZZAZIONE DATI
2 '
3 ' Questo programma considera che il Disk o l'Advanc
ed BASIC siano
4 ' stati avviati con i parametri di default per le
opzioni /S: e /C:
5 '
```

(continua)

```
6 '
10 SCREEN 0,0:KEY OFF
20 XON$=CHR$(17): XOFF$=CHR$(19) : SOSPENDI=0
30 OPEN "ANALIZZA.DAT" FOR OUTPUT AS #2 ' Il file seq
uenziale "ANALIZZA.DAT"
35 ' conterra' i dati in uscita dallo strumento
40 OPEN "COM1:1200,N,8" AS #3 ' Apre il file #3 come
linea di comunicazione
45 ' senza parita', con 8 bit di dati per parola e 1
bit di stop
50 ON COM(1) GOSUB 480 ' Quando ci sono dati in input
 li elabora
60 GOSUB 390 ' Visualizza il menu dei comandi
70 INPUT C: ON C GOTO 110,150,230,190 ' Scelta dell '
input, salta alla
75 ' routine dei comandi
80 PRINT "CATTIVA SCELTA, PROVA ANCORA" ' Messaggio
d'errore
90 FOR Q=0 TO 500:NEXT
100 GOTO 60
110 PRINT #3, "A0" ' "A0" e' il comando di inizializ
zazione
120 PRINT "STRUMENTO INIZIALIZZATO"
130 FOR Q=0 TO 500:NEXT
140 GOTO 60
150 PRINT #3, "A1" ' "A1" e' il comando che spegne lo
 strumento
160 PRINT "STRUMENTO SPENTO"
170 FOR Q=0 TO 500:NEXT
180 GOTO 60
190 PRINT #3, "A1" ' "A1" e' il comando che spegne lo
 strumento
200 PRINT "STRUMENTO SPENTO E PROGRAMMA TERMINATO"
210 FOR Q=0 TO 500:NEXT
220 CLOSE:END
230 CLS
240 LINE INPUT "INSERIRE DATI DI DUE CIFRE; ES. 39,00
,32 <ENTER> - ";COMDAT$
250 PRINT #3,"A2" ' "A2" e' il comando di attesa
260 PRINT " DATI STRUMENTALI - "
270 FOR Q=0 TO 500 :NEXT
280 Y=0
290 FOR X=1 TO (LEN(COMDAT$)+1)/3
300 Y=Y+1
310 Z$=MID$(COMDAT$,Y,2) ' Rileva il dato di 2 cifre
320 PRINT VAL(Z$) ' Visualizza il valore numerico del
 dato
330 PRINT  #3, VAL(Z$) ' Invia il valore del dato
```

(continua)

```
340 Y=Y+2  ' Incrementa il contatore perche' punti al
 dato successivo
350 NEXT
360 PRINT #3, "A3" ' "A3" e' il comando di termine da
ti
370 FOR Q=0 TO 900:NEXT
380 GOTO 60
390 CLS
400 PRINT "INSERIRE LA SCELTA DEL COMANDO (1-4) - "
410 PRINT
420 PRINT "1. INIZIALIZZA LO STRUMENTO"
430 PRINT "2. SPEGNI LO STRUMENTO"
440 PRINT "3. INVIA DATI DI COMANDO"
450 PRINT "4. SPEGNI E TERMINA"
460 PRINT
470 RETURN
480 IF LOC(3)>128 THEN SOSPENDI=1:PRINT #3,XOFF$ ' Se
 il buffer di input e'
485 ' pieno piu' che per meta', definisce il flag SOS
PENDI e sospende
486 ' l'input con il carattere XOFF
490 A$=INPUT$(LOC(3),#3) ' Pone il contenuto del buff
er di input in A$
500 FOR I=1 TO LEN(A$) ' Controlla il codice d'errore
 per lo strumento
510 IF MID$(A$,I,1)=CHR$(27) THEN 550 ' Se riceve il
carattere ESCAPE
513 ' (codice ASCII 27) allora va alla routine d'erro
re
515 NEXT
520 PRINT #2,A$: IF EOF(3) THEN 530 ELSE 480 ' Memori
zza i dati sul disco
525 ' Se l'input non e' finito riprende
530 IF SOSPENDI=1 THEN SOSPENDI=0:PRINT #3,XON$ ' Rip
rende l'input dello
535 ' strumento con il carattere XON
540 RETURN
550 PRINT "ERRORE DELLO STRUMENTO"
560 GOTO 150 ' Spegne lo strumento e torna al menu pr
incipale
```

**Figura 11.10** Programma per il controllo di uno strumento e la memorizzazione di dati

Capitolo

# Comandi DOS per lo sviluppo dei programmi

# 12

Sul disco del DOS si trovano diversi comandi progettati specificatamente per fornire un aiuto nel processo di sviluppo dei programmi; questi comandi comprendono EDLIN, un editor, LINK, un linker di programmi e DEBUG, uno strumento per trovare errori nei programmi.

EDLIN viene usato per creare file, comporli e modificarli: si comporta, cioè, come un semplice word processor e potete usarlo per creare file di tipo testo: se vi interessano, però, le prestazioni complete di un word processor, vi conviene utilizzare un programma più sofisticato.

L'uso principale di EDLIN è quello di creare file "sorgente", cioè file che contengono le istruzioni non ancora compilate dei vari programmi. Questi file vengono poi usati come input per un programma assemblatore o compilatore al fine di creare file "oggetto", cioè file contenenti la versione in codice macchina dei vostri programmi.

LINK viene usato nel procedimento di creazione di un programma finale eseguibile. LINK prende un insieme di file oggetto e crea un file "eseguibile", cioè un file che può essere mandato in esecuzione sul PC. Questo non serve con il BASIC normale, ma è necessario con il BASIC compilato, il Pascal, il Macro Assembler, il FORTRAN e così via.

DEBUG, invece, viene usato per cercare gli errori nei programmi in linguaggio Assembler: vi permette cioè di caricare ed eseguire programmi Assembler controllandoli passo passo. Con DEBUG, potete fermare l'esecuzione del programma da controllare per esaminare e, se necessario, modificare le istruzioni.

DEBUG, inoltre, vi fornisce un modo per esaminare e modificare i file e vi permette di vedere un file esattamente come è memorizzato nel PC.

Per chi è interessato allo sviluppo dei programmi il disco del DOS contie-

ne quindi molti utili strumenti. Altri strumenti, comunque, come l'Assembler 8086/8088 o il compilatore per un linguaggio di alto livello come FORTRAN, Pascal o BASIC, devono essere acquistati separatamente ed usati insieme con il DOS e con EDLIN, LINK, e DEBUG.

## 12.1 EDLIN

EDLIN crea file composti da un insieme di linee di non più di 253 caratteri ciascuna; è simile a un programma BASIC, infatti ad ogni linea di un file creato con EDLIN viene assegnato un numero di linea a cui fanno riferimento tutti i comandi di EDLIN e ciò vi permette di spostarvi a piacere all'interno del file.
I numeri di linea che vedete, però, non vengono inseriti come parte del file; in altre parole, se stampate un file creato con EDLIN, i numeri di linea non compaiono.
EDLIN può essere usato sia per comporre file di testo che programmi in formato sorgente; potreste creare un file contenente un modulo di lettera, memorizzarlo su disco e poi modificarlo a seconda delle necessità, con EDLIN.
Ripetiamo, però, che questo programma non è adatto per un lavoro esteso di word processing, perché non fornisce gli strumenti necessari per una gestione completa del testo.

### USO DI EDLIN

Per mandare in esecuzione EDLIN dovete inserire nel drive corrente il disco contenente il DOS e poi battere il seguente comando in risposta al prompt:

EDLIN *specfile* [/B]

Il parametro *specfile*, nel comando EDLIN, deve riferirsi ad un file su disco; se manca lo specificatore del drive, viene utilizzato il drive di default. *specfile* può contenere anche il cammino di un directory, come spiegato nel Capitolo 3.

NOTA: il parametro facoltativo /B permette a EDLIN di superare un carattere EOF (CTRL Z) all'interno del file. Inoltre, assicuratevi di avere sufficiente spazio sul disco per contenere la nuova versione del file, poiché la vecchia verrà conservata come copia di backup con l'esten-

sione .BAK; tutto o parte del vostro nuovo file verrà irrimediabilmente perso se il disco è pieno quando chiudete EDLIN.

Dopo aver caricato EDLIN, il DOS cerca il file da voi indicato: se lo trova, lo carica nella memoria del PC finché questa non sia piena al 75%; il resto dello spazio viene riservato per contenere le eventuali aggiunte. Se invece il file non viene trovato, EDLIN assume che voi stiate creando un nuovo file.
Se il file viene trovato e può essere caricato completamente in memoria, EDLIN vi risponde con:

```
End of input file
*
```

mentre se il file è nuovo con:

```
New file
*
```

Se invece il file che vi interessa è molto ampio e non può essere tutto contenuto nella memoria, EDLIN ne carica quanto possibile e risponde con:

```
*
```

L'asterisco da solo indica appunto che il file è troppo grande per essere caricato completamente nella memoria. Vedremo più avanti come caricare anche la parte del file rimasta fuori dalla memoria.
In ogni caso ricordate che l'asterisco è il prompt di EDLIN: ogni volta che ne appare uno, sapete che EDLIN si aspetta un comando o una linea di testo da inserire nel file. Vediamo ora come utilizzare la tastiera con il programma EDLIN.

## USO DELLA TASTIERA CON EDLIN

Il DOS supporta alcune funzioni avanzate in grado di gestire la redazione di una singola linea; descriveremo queste funzioni nell'ambito della discussione su EDLIN. Notate però che tutte queste funzioni si riferiscono in generale al DOS e potete sempre usarle nell'invio dei comandi DOS.

## Uso del template

Le funzioni di correzione del DOS sono basate sulla manipolazione di una sola linea di testo per volta: ogni volta che viene premuto il tasto ENTER, viene inviata al PC una linea completa.
Il DOS mantiene una copia delle linee che voi inviate in un buffer chiamato *template*, che vi permette di apportare cambiamenti alle linee senza doverle ribattere completamente.
Normalmente, il template contiene l'ultima linea che voi avete inviato; se, in risposta al prompt del DOS (o di EDLIN) battete semplicemente una nuova linea, magari correggendola con il tasto BACKSPACE e poi battendo ENTER, il template conterrà una copia della linea appena immessa. Utilizzando i tasti funzione potete avere accesso al template e trasferire parte del contenuto dal template allo schermo.

## I tasti di correzione

I tasti elencati nella Tabella 12.1 permettono di trasferire parti di testo dal template alla linea visualizzata e di manipolare quest'ultima.
Per creare un nuovo file inserite una copia del disco con il DOS nel drive A: e richiamate EDLIN in questo modo:

```
EDLIN TESTOUNO.TXT
```

In risposta al prompt di EDLIN scrivete la lettera I. Questo comando di inserimento ordina a EDLIN di inserire tutto il testo nel file che stiamo creando. In questo momento il file è, in effetti, vuoto. Sullo schermo compaiono queste scritte:

```
New file
*I
1:*_
```

Il simbolo 1:* significa che EDLIN porrà tutto ciò che state per battere da tastiera nella linea numero 1 del nuovo file.
Ora battete una breve linea, come questa, e poi inviatela al PC:

```
1:*Mi piacerebbe sapere perché sono così stonato
2:*_
```

EDLIN ha posto ora la linea visualizzata (senza il numero di linea) nella prima linea del nuovo file. Il simbolo 2:* significa che la prossima linea visualizzata che verrà inviata, diverrà la seconda linea del nuovo file.

**Tabella 12.1** Tasti di correzione

| Tasto | Funzione |
|---|---|
| ENTER | Invia la linea visualizzata al programma corrente ed al template |
| BACKSPACE | Cancella un carattere dalla linea visualizzata e sposta il cursore di una posizione a sinistra. Il template non viene modificato |
| ESC | Cancella la linea visualizzata e abbassa il cursore di una linea in modo da lasciarvi l'opportunità di riscrivere una nuova linea. Il template rimane invariato |
| F1 (o cursore a destra) | Trasferisce un carattere dal template alla linea visualizzata e sposta di una posizione verso destra il cursore del template |
| DEL | Muove il cursore del template di una posizione verso destra; la linea visualizzata non cambia |
| F2 *carattere* | Trasferisce nella linea visualizzata il contenuto del template a partire dalla posizione corrente del cursore fino al carattere indicato |
| F3 | Trasferisce tutti i caratteri dal template alla linea visualizzata, a partire dalla posizione del cursore del template |
| F4 *carattere* | Come la funzione F2 *carattere*, eccetto che la linea visualizzata non viene cambiata |
| F5 | Trasferisce la linea visualizzata al template e non influisce sul programma corrente |
| INS | Il cursore del template resta fermo mentre vengono inseriti nuovi caratteri |

Vediamo ora cosa si trova nel template. Premete F3 per far comparire il contenuto di questo nella linea numero 2; vedrete comparire:

```
2:*Mi piacerebbe sapere perché sono così stonato_
```

Il template viene visualizzato a partire dalla posizione corrente del cursore, che in questo caso è l'inizio della seconda linea.
Per iniziare un'ulteriore linea, premete F5, che ha come effetto:

```
2:*Mi piacerebbe sapere perché sono così stonato@
   _
```

Osservate che il cursore sul video si è spostato alla riga successiva. Il contenuto del template non è cambiato perché F5 invia una copia della linea visualizzata al template; osservate che invece questa linea non è stata inviata a EDLIN.
Ora vogliamo trasferire una parte del template alla linea appena cancellata. Innanzitutto premete F1 (o cursore a destra) per sette volte e vedrete apparire:

```
2:*Mi piacerebbe sapere perché sono così stonato
   Mi piac_
```

I primi sette caratteri del template sono stati riportati sulla linea visualizzata ed i due cursori, quello del video e quello del template si sono mossi di conseguenza.
Ora riportate indietro entrambi i cursori con il tasto BACKSPACE in questo modo:

```
2:*Mi piacerebbe sapere perché sono così stonato
   Mi pia_
```

Dalla linea visualizzata è stato cancellato un carattere ed i due cursori si sono spostati di una posizione verso sinistra; il contenuto del template, però, non è stato modificato. Per muovere il cursore del template di un carattere a destra ed inviare un carattere da questo allo schermo, premete di nuovo F1.

```
2:*Mi piacerebbe sapere perché sono così stonato
   Mi piac_
```

Ora facciamo avanzare il cursore fino alla successiva occorrenza della lettera "é", copiando i caratteri che compaiono fino a questa lettera nella linea visualizzata, per mezzo del tasto di funzione F2 seguito dalla lettera "é". Viene così visualizzato il resto del template fino alla parola "perché", che la contiene:

```
2:*Mi piacerebbe sapere perché sono così stonato
   Mi piacerebbe sapere perch_
```

Se in questo punto volete inserire altri brani di testo, premete il tasto INS: in questo modo, qualunque carattere voi battiate andrà a far parte della linea visualizzata, ma il cursore del template non si sposterà. Se premete di nuovo INS, il nuovo testo andrà a coprire quello già presente sullo schermo ed anche il cursore del template si muoverà in accordo.
Se volete oltrepassare alcuni caratteri nel template, premete il tasto DEL,

che fa spostare il cursore del template di una posizione verso destra, cancellando in realtà il carattere dal template.
Infine, se volete ricominciare da capo dopo aver iniziato una nuova linea, premete il tasto ESC: un backslash ( \ ) appare al termine della linea visualizzata ed il cursore dello schermo si sposta alla riga successiva, mentre il template resta invariato.
Abbiamo fin qui mostrato le funzioni di editing da tastiera per una singola linea all'interno di EDLIN; ricordate che queste funzioni sono proprie del DOS e hanno gli stessi effetti anche all'esterno di EDLIN. Potete utilizzare tutte queste funzioni, compreso il template appena descritto, nell'invio dei comandi DOS.

## I COMANDI EDLIN

In risposta al prompt di EDLIN, si possono dare numerosi comandi oltre a I che abbiamo appena spiegato.
Ogni comando EDLIN è formato da un singolo carattere, (non importa se maiuscolo o minuscolo) con o senza parametri. Un comando EDLIN viene eseguito al momento in cui premete il tasto ENTER e può essere fermato premendo i tasti CTRL BREAK, che vi restituiscono il prompt dell'EDLIN. Premendo invece CTRL NUM LOCK si sospende temporaneamente l'output di un comando in modo che voi possiate, ad esempio, esaminarlo agevolmente; premendo un altro qualsiasi tasto fate riprendere l'esecuzione del comando interrotto.

**Tabella 12.2** I parametri per i comandi EDLIN

| Parametro | Definizione |
|---|---|
| *linea* | Il parametro *linea* si riferisce ai numeri di linea nel file creato dal programma editor e può essere: un intero compreso tra 1 e 65529, un punto (.) per indicare la linea corrente, un simbolo # per indicare la linea che segue l'ultima in memoria. Se utilizzate un numero di linea maggiore del numero di linee effettivamente contenute nel file ottenete un effetto identico a quello del parametro # |
| *n* | Indica il numero delle linee che devono essere trasferite dal disco in memoria o viceversa; il parametro viene utilizzato con i comandi di aggiunta e scrittura quando il file risulta troppo grosso per essere tutto contenuto in memoria |
| *stringa* | Rappresenta una stringa di caratteri che il programma EDLIN usa come modello nel comando di ricerca e sostituzione |

I delimitatori (virgola o spazio bianco) sono necessari solo quando due numeri di linea vengono usati come parametri per un comando; altrimenti vengono ignorati. I parametri per i comandi EDLIN vengono presentati nella Tabella 12.2.
I comandi EDLIN vi permettono di inserire linee in qualunque punto del file, di esaminare un determinato gruppo di linee, di cancellarne, di ricercare e sostituire brani di testo e di riscrivere il file corretto sul disco originale. A questo punto vi trovate proprio nel mezzo della composizione del file "TESTOUNO.TXT". Provate ad eseguire su questo file i numerosi comandi prima presentati per farvi un'idea di come funzionino; potete usare comandi multipli su di una sola linea, purché li separiate con un punto e virgola.

### Comando inserimento linee - I

Il comando I permette di inserire linee di testo in un punto qualsiasi del file. La sintassi del comando è la seguente:

[*linea*] I

Se viene omesso il parametro *linea*, le linee di testo inserite vengono aggiunte in coda alla linea corrente; se invece viene indicato un numero di linea, le nuove linee vengono inserite prima di quella indicata fino al termine del comando. Premendo il tasto ENTER inviate la linea visualizzata al file e provocate la visualizzazione del successivo numero di linea da parte di EDLIN.
Il comando di inserimento linee termina quando voi battete CTRL BREAK.
Quando le linee vengono inserite nel file, tutte le successive vengono rinumerate.

### Comando correzione linea

Il comando di correzione linea recupera una linea dal file perché possa essere modificata. La sintassi è semplicemente:

[*linea*]

Se viene indicato un numero di linea, quella linea viene visualizzata, e potete utilizzare i consueti tasti di correzione per modificarla oppure semplicemente potete riscriverla.
Se invece non viene indicato alcun numero di linea, se premete cioè ENTER in risposta al prompt dei comandi EDLIN, viene visualizzata la linea successiva a quella in cui vi trovate.

Se non volete cambiare la linea dopo averla recuperata, premete ENTER quando il cursore dello schermo si trova all'inizio di questa; anche premendo ESC o CTRL BREAK lasciate invariata la linea che si trova nel file. Se volete inserire dei caratteri di controllo all'interno del file-testo, tutto quello che dovete fare è battere CTRL V seguito da una lettera maiuscola: ad esempio, CTRL V Z inserisce il carattere Z nel file.

### Comando visualizzazione linee - L

Il comando L è utilizzato per visualizzare un gruppo di linee del file. La sintassi è la seguente:

[*linea*] [,*linea*] L

Se non vengono inclusi parametri, vengono visualizzate 23 linee (centrate attorno alla linea corrente). Se viene omesso il primo, cioè se battete:

,*linea* L

la visualizzazione parte da 11 linee prima di quella corrente e termina con la linea indicata.
Se invece definite solo il primo parametro, cioè se battete:

*linea* L

vengono visualizzate 23 linee, a partire da quella indicata.
Se invece vengono specificati entrambi i parametri, viene visualizzato l'intero intervallo richiesto.

### Comando pagina - P

Il comando pagina (P) è identico al comando L, con l'eccezione che il numero di linea corrente viene modificato. La sintassi del comando è:

[*linea*] [,*linea*] P

L'ultima linea visualizzata dopo il comando P diventa la linea corrente; il comando è utile per muoversi lungo il testo visualizzando una pagina (23 linee) per volta.

### Comando copiatura linee - C

Il comando C produce una copia di una o più linee di testo e la pone nel file, nel punto indicato. La sintassi di questo comando è la seguente:

[*linea*],[*linea*],*linea* [,*contatore*]C

Sia il primo che il secondo parametro assumono il valore della linea corrente se non vengono specificati, altrimenti indicano l'insieme di linee da copiare. Il terzo parametro indica il primo numero di linea da assegnare alle linee copiate. Se, per esempio, volete copiare le linee dalla 1 alla 6 nella linea 10, le linee da 1 a 6 vengono riprodotte esattamente nelle linee dalla 10 alla 15 e quella che era la linea 10 diviene la linea 16.
Se viene incluso anche l'ultimo parametro, l'operazione viene ripetuta il numero di volte indicato in *contatore*; se invece viene omesso, l'operazione viene effettuata una volta sola.

### Comando cancellazione linee - D

Il comando D cancella una o più linee di testo; la sintassi è:

[*linea*] [,*linea*]D

I due parametri indicano l'intervallo di linee da cancellare. Il primo assume per default il valore della linea corrente, mentre solo la linea indicata viene cancellata se il secondo viene omesso. Al termine dell'operazione, la linea corrente diviene quella che seguiva immediatamente l'ultima linea cancellata.

### Comando spostamento linee - M

Il comando M sposta una o più linee dalla posizione corrente ad una nuova locazione. La sua sintassi è la seguente:

[*linea*],[*linea*],*linea* M

I primi due parametri indicano l'insieme di linee da muovere e per default rappresentano la linea corrente. Il terzo parametro indica il numero di linea a cui deve essere trasferito il blocco identificato; in pratica, se muovete le linee dalla 10 alla 12 nella linea 3, avvengono diversi mutamenti: le linee mosse scompaiono dalla loro posizione originale e riappaiono nella posizione indicata ed i numeri di linea si modificano in ac-

cordo con il cambiamento, in modo che le linee che avevano i numeri da 3 a 6 diventano le linee alla 9 alla 12.

### Comando ricerca testo - S

Il comando S viene utilizzato per trovare le linee che contengono una specifica sequenza di caratteri. La sintassi del comando è:

[*linea*] [,*linea*] [?] S*stringa*

L'intervallo di linee tra cui svolgere la ricerca può essere definito assegnando i valori per i primi due parametri. Il parametro *stringa* non deve essere racchiuso tra virgolette a meno che queste facciano parte dei caratteri da identificare. Se il primo parametro di linea viene omesso, assume per default il valore 1; se viene omesso il secondo, invece, viene utilizzata per default l'ultima linea del file; tralasciando entrambi i parametri, la ricerca avverà lungo tutto il file.
Se viene omesso il simbolo ?, la ricerca ha termine dopo che è stata trovata la prima linea in cui compare la stringa indicata, e questa linea diviene la linea corrente. Se invece la linea non viene trovata, il comando termina e visualizza il messaggio NOT FOUND.
Aggiungendo anche il simbolo ?, EDLIN mostra ogni volta sia la linea trovata che il messaggio "O.K.?". Se premete il tasto Y o ENTER, la ricerca termina e la linea trovata diviene la linea corrente, altrimenti, premendo un altro qualsiasi tasto, la ricerca riprende.
Osservate che la stringa da cercare è formata da tutti i caratteri che compaiono nel comando alla destra di S, fino al punto in cui viene premuto ENTER.

### Comando sostituzione testo - R

Il comando R funziona come il precedente, tranne per il fatto che la stringa indicata viene sostituita da un'altra che voi avete inserito come parte del comando.
La sintassi del comando R è:

[*linea*] [,*linea*] [?] R*stringa* [<F6> *stringa*]

L'intervallo di linee in cui cercare l'occorrenza del primo parametro *stringa* è definita in modo identico a quello del comando S. Se viene aggiunto il simbolo ?, la ricerca viene sospesa ogni volta che la prima stringa è trovata all'interno di una linea; battendo Y o ENTER in risposta al

messaggio "O.K.?" la stringa trovata viene sostituita dal secondo parametro indicato, mentre battendo un altro qualsiasi la ricerca riprende senza modificare la linea corrente. In entrambi i casi la ricerca continua fino alla fine dell'intervallo indicato.
Il primo parametro *stringa* viene incluso appena dopo la R e termina quando premete il tasto F6; in modo analogo la stringa sostitutiva è battuta subito dopo il tasto F6 e termina quando premete ENTER. Se non viene indicata alcuna stringa sostitutiva, la prima stringa viene semplicemente cancellata ogni volta che viene identificata.

### Comando trasferimento linee - T

Il comando T legge il contenuto di un file su disco e lo fonde con il file corrente. La sintassi del comando T è la seguente:

[*linea*]T[*d:*] *nomefile* [*.est*]

Il parametro *linea* indica la posizione delle nuove linee di testo e se omesso assume per default il valore della linea corrente; va inoltre indicato il drive e il nome del file.

### Comando scrittura - W

Il comando W viene usato quando un file è troppo grande per essere contenuto nella memoria del PC, cioè nel caso in cui, dopo la chiamata del programma EDLIN non ottenete uno di questi messaggi:

```
End of input file
*

New file
*
```

Il file viene caricato in memoria fino a riempirla per il 75%. La parte di file caricata può essere manipolata a piacere e quando avete terminato con questa, dovete usare il comando W per trasferire il materiale modificato di nuovo sul disco allo scopo di lasciare spazio in memoria per il resto del file. (Vedi il paragrafo che segue per la lettura del nuovo testo dal disco).
La sintassi di questo comando è:

[*n*] W

Se *n* viene omesso, tutte le linee del file, a partire dalla numero 1, vengono scritte sul disco fino a che la memoria non è occupata per il 25%; se invece *n* viene specificato, vengono scritte sul disco le linee dalla 1 alla *n*.

**Comando aggiunta linee - A**

Dopo aver usato il comando W per liberare spazio nella memoria, utilizzate il comando A per caricare la parte successiva del file. La sintassi di A è questa:

[*n*] A

Se *n* non viene indicato, le linee vengono trasferite nella memoria del PC fino a quando questa non è piena per il 75% o quando viene raggiunta la fine del file. Specificando *n*, invece, solo *n* linee di testo verranno aggiunte all'ultima linea in memoria.
Se tutta la parte restante del file è stata caricata per mezzo del comando A, viene visualizzato il messaggio END OF INPUT FILE (fine del file di input).

**Comando di chiusura - E**

Il comando E determina la scrittura del file modificato sul disco: fino a quando non inviate questo comando, infatti, tutte le modifiche vengono fatte solo nella memoria del PC. Con il comando E, il file originale viene chiamato con lo stesso nome e con la nuova estensione .BAK. Il file modificato, invece, appena salvato sul disco, assume l'intera specificazione originale. Se nel drive corrente si trova un nuovo disco, il file modificato viene scritto con la stessa specificazione indicata nel comando EDLIN.
La sintassi di questo comando è semplicemente:

E

Di nuovo, dovete assicurarvi che nel disco in cui volete scrivere il file ci sia abbastanza spazio, altrimenti una parte del file modificato andrà persa irrimediabilmente.
Il comando E conclude il lavoro di EDLIN e vi riporta al DOS.

### Comando di uscita - Q

Il comando Q termina la sessione corrente di EDLIN senza salvare il file modificato. Anche in questo caso la sintassi è semplice:

Q

Prima di abbandonare effettivamente il file, EDLIN vi chiede conferma della richiesta con il messaggio:

```
Abort edit (Y/N)?
```

Battendo il tasto Y abbandonate EDLIN mentre un altro qualsiasi tasto vi riporta il prompt di EDLIN.
Se utilizzate il comando Q, non viene creata alcuna copia di backup del file modificato, ma l'originale rimane immutato.

## 12.2 LINK

LINK lavora su file oggetto creati da un programma assemblatore o da un compilatore e produce un file eseguibile dal PC.
I tipi di file che LINK può elaborare sono elencati nella Tabella 12.3. I file *Libreria* contengono subroutine e funzioni a cui si può far riferimento da altri programmi: questi file vengono normalmente creati da un compilatore.

**Tabella 12.3** File di input per LINK

| File di input | Estensione | Generato da: |
|---|---|---|
| Oggetto | .OBJ | Assemblatore |
| Oggetto | .OBJ | Compilatore |
| Libreria | .LIB | Compilatore |

**Tabella 12.4** File di output di LINK

| File di output | Estensione | Usato da: |
|---|---|---|
| Eseguibile | .EXE | COMMAND.COM |
| List | .MAP | Utente |
| (Temporaneo) | .TMP | LINK |

I file prodotti da LINK sono elencati nella Tabella 12.4; il primo file è un file eseguibile (con estensione .EXE) che il DOS può caricare in memoria e poi mandare in esecuzione; il secondo è un file di list (con estensione .MAP), che mostra dove i vari programmi mandati in input a LINK sono stati posti nel programma eseguibile.

## USO DI LINK

Per usare LINK dovete inviare il seguente comando al DOS:

LINK [*specfile*]

Il parametro opzionale *specfile* può contenere un file di *risposta automatica*, cioè un file molto simile ad un file batch. Questo file, infatti, che può essere creato per mezzo di EDLIN, contiene le risposte ai prompt che LINK genera da quando viene avviato. Tra breve esamineremo per maggior chiarezza un esempio.
Se non viene indicata la specificazione, LINK risponde con una serie di prompt, elencati con le opportune risposte nella Tabella 12.5; i parametri opzionali, invece, sono riportati nella Tabella 12.6.

**Tabella 12.5** Parametri di LINK

| Prompt | Risposta |
|---|---|
| OBJECT MODULES | Una o più specificazioni di file, separate da uno spazio o da una virgola, che indicano i file contenenti i programmi che dovranno comporre il programma finale eseguibile. Se uno dei file indicati non viene trovato, LINK ve lo richiede, permettendovi di tenere file di input su diversi dischi |
| RUN FILE | Il nome del file eseguibile che LINK deve produrre. Se la specificazione viene seguita da P, LINK, quando è pronto, vi chiede di inserire il disco nel quale dev'essere immagazzinato il file eseguibile |
| LIST FILE | Se viene indicata una specificazione, verrà creato e memorizzato sotto questa stessa specificazione un file list |
| LIBRARIES | Nome/i di una o più librerie da usare |
| OPTIONAL PARAMETERS | /D, /H, /L, /M, /P, /S dopo ogni risposta a uno dei prompt precedenti |

**Tabella 12.6** Parametri opzionali di LINK

| Parametro | Nome | Funzione |
|---|---|---|
| /D | DSALLOCATION | Carica tutti i dati definiti come parte del DGROUP nella parte più alta del gruppo |
| /H | HIGH | Pone il file eseguibile il più in alto possibile nella memoria |
| /L | LINE | Aggiunge i numeri di linea del programma sorgente e gli indirizzi nel file list |
| /M | MAP | Elenca tutti i simboli globali, i loro valori ed offset nel file list |
| /P | PAUSE | Sospende l'esecuzione per consentire l'inserimento del dischetto con il file eseguibile |
| /S:*dimensione* | STACK | Definisce la dimensione dello stack, da 512 a 65536 byte |

## COME CREARE UN FILE DI RISPOSTA AUTOMATICA

Un file di risposta automatica può essere usato per rispondere alle numerose richieste di LINK. Il formato del file è tale che ogni linea corrisponde ad una risposta ad un singolo prompt.
Supponiamo di dover dare le seguenti risposte per i prompt di LINK:

```
A>LINK

IBM Personal Computer Linker
Version 1.10 (C)Copyright IBM Corp 1982

Object Modules [.OBJ]: PRIMOMOD.OBJ+SECMOD.OBJ+CALC.OBJ
Run File [PRIMOMOD.EXE]: GO
List File [NUL.MAP]: GOLIST
Libraries [.LIB]: PASCAL.LIB
```

Per simulare queste risposte, possiamo creare un file di risposta automatica con le seguenti linee:

```
PRIMOMOD.OBJ+SECMOD.OBJ+CALC.OBJ
GO
GOLIST
PASCAL.LIB
```

La prima linea di questo file indica i tre file di input per LINK; il resto del file contiene le risposte per gli altri prompt.

## 12.3 DEBUG

Lo strumento più idoneo per la ricerca degli errori nei programmi scritti in Assembler è DEBUG, che vi mette a disposizione alcuni comandi per controllare ed esaminare ogni passo del programma in esecuzione.

**Tabella 12.7** I comandi di DEBUG

| Comando | Funzione |
|---|---|
| A - Assemble | Assembla i codici mnemonici direttamente nella memoria |
| C - Compare | Confronta il contenuto di due blocchi di memoria e visualizza le differenze trovate |
| D - Dump | Visualizza il contenuto di un blocco di memoria sullo schermo, sia in esadecimale che in formato ASCII |
| E - Enter | Invia o modifica il contenuto della memoria |
| F - Fill | Riempie ogni byte di un blocco di memoria fino al limite fissato del parametro |
| G - Go | Inizia l'esecuzione del programma che deve essere controllato con DEBUG |
| H - Hexarithmetic | Somma o sottrae due numeri esadecimali e visualizza il risultato |
| I - Input | Acquisisce il valore corrente di una porta di input |
| L - Load | Carica il contenuto di un file o di un blocco di settori di un disco in memoria |
| M - Move | Sposta il contenuto di un blocco di memoria ad una specificata locazione |
| N - Name | Definisce la specificazione ed i parametri da usare per l'input e l'output dei dischi |
| O - Output | Invia un byte ad una determinata porta di output |
| Q - Quit | Lascia DEBUG e ritorna al DOS |
| R - Register | Visualizza e modifica i registri interni |
| S - Search | Ricerca in un blocco di memoria una serie di valori |
| T - Trace | Esegue una o più istruzioni, visualizzando il contenuto dei registri e lo stato dei flag |
| U - Unassemble | Disassembla il contenuto della memoria in codici mnemonici |
| W - Write | Scrive dati su disco |

DEBUG è usato di solito nello sviluppo di un nuovo programma di cui volete controllare il funzionamento a livello di codice macchina. Per usare DEBUG dovete avere un listato in Assembler del programma, oltre ad una buona conoscenza del microprocessore Intel 8088. Nella Tabella 12.7 trovate un elenco dei comandi di DEBUG.
Proponiamo ora una semplice dimostrazione dell'uso di DEBUG, tanto per vedere cosa è in grado di realizzare.

## UNA BREVE DIMOSTRAZIONE DI DEBUG

Per richiamare DEBUG basta il comando:

```
DEBUG BASICA.COM
```

che indica al DOS di caricare DEBUG insieme al file di comandi BASICA.COM, l'interprete dell'Advanced BASIC. Un trattino (−) è il prompt di DEBUG: quando questo compare, sapete che il programma è in attesa di un comando interno di DEBUG.
Vediamo ora come appare l'interprete Advanced BASIC in memoria; inviamo il comando Dump:

```
D
```

Quello che appare sullo schermo è il contenuto di un segmento della memoria del PC, dall'offset 0100H all'offset 017FH. Ogni coppia di caratteri rappresenta il valore esadecimale di una data locazione di memoria. Inoltre, nella colonna a destra dell'elenco compaiono i caratteri corrispondenti al contenuto della memoria se interpretato come codice ASCII: questa "traduzione" è molto utile quando si esamina un file in formato ASCII. Qui, invece, l'interpretazione ASCII è senza significato, poiché questa zona di memoria contiene il codice macchina dell'interprete Advanced BASIC.
Esaminiamo ora l'Assembler equivalente al contenuto di memoria appena visto; inviamo il comando:

```
U
```

che, partendo dal codice macchina, lo converte di nuovo in Assembler. Quello che vedete ora sullo schermo è la prima linea dell'interprete Advanced BASIC nella sua forma in Assembler.
Osservate che l'indirizzo iniziale in questo elenco disassemblato è lo stesso di quello in codice macchina.

Eseguiamo quindi Advanced BASIC sotto DEBUG e per questo battiamo il comando:

```
G
```

Appare subito la videata tipica dell'Advanced BASIC: osservate il valore fornito in seguito al messaggio NUMBER OF BYTES FREE: questo valore è minore di quello che normalmente appare quando l'Advanced BASIC è avviato direttamente dal DOS. Questa riduzione di spazio è dovuta al posto che DEBUG occupa in memoria.
Per ritornare a DEBUG inviate il comando:

```
SYSTEM
```

Appare il messaggio PROGRAM TERMINATED NORMALLY, insieme al prompt. Per uscire da DEBUG, usate il comando Quit:

```
Q
```

Capitolo

# Manutenzione del PC

# 13

Il PC-IBM pur essendo uno strumento estremamente complesso ha una notevole affidabilità e funziona per la maggior parte del tempo senza inconvenienti.
Se vi imbattete in un guasto, però, non dovrete sempre coinvolgere degli esperti per individuarlo e ripararlo.
Questo capitolo vi spiegherà come trattare il PC in modo da evitare la maggior parte dei problemi, come intervenire se qualcosa non funziona e dove trovare un ulteriore aiuto, quando da soli non foste in grado di risolvere ogni cosa.

## 13.1 Come avere cura del computer

L'affidabilità del funzionamento del PC dipende da come il sistema è stato installato e da come è alimentato. Una volta messo in funzione, corrette procedure operative evitano la maggior parte dei problemi.

### COME INSTALLARE IL SISTEMA

Dove sistemare il vostro nuovo computer e tutti gli accessori? La soluzione ideale è di avere molto spazio per disporre con agio il PC e tutti i suoi componenti.
Quando installate il PC, evitate di ammassarne i componenti: la maggior parte di questi, infatti, richiede un notevole flusso d'aria intorno per un

buon raffreddamento. Non esponete mai il PC alla luce diretta del sole che contribuisce a scaldarlo ulteriormente: l'affidabilità del sistema ne può soffrire. La temperatura dell'ambiente in cui lavorate dovrebbe essere compresa tra 15 e 25 gradi centigradi.
Inoltre, la stanza in cui tenete il computer dovrebbe essere un ambiente ben ventilato e pulito; ogni fonte di polvere e sporcizia dovrebbe essere tenuta sotto controllo in modo da non contaminare i componenti del sistema ed in particolare i drive dei dischi.
Scariche di elettricità statica, come quelle scintille che scoccano quando toccate una superficie metallica dopo aver camminato su un tappeto in una giornata molto secca, possono rovinare il computer; per prevenire queste scariche dovete mantenere l'umidità relativa al di sopra del 50%, non usare tappeti sintetici nella stanza in cui tenete il PC ed utilizzare stuoie o spray antistatici.
La comodità dell'operatore è molto importante, soprattutto quando il computer viene utilizzato per lunghi periodi. Dovete installare lo schermo in modo da poterlo vedere senza sforzo o fatica. La tastiera deve trovarsi nel posto che per voi risulta più comodo: se volete utilizzarla tenendola sulle ginocchia, vi servirà senz'altro un cavo di collegamento più lungo. Assicuratevi che i drive siano facilmente accessibili, per ridurre il rischio di danni ai dischi quando li inserite o li estraete dal drive.

## ALIMENTAZIONE DEL PC

Collegare ad una presa un computer non è lo stesso che collegare un elettrodomestico o un amplificatore stereo, perché il computer ha pretese molto più vincolanti per l'alimentazione esterna. In particolare, il PC è molto sensibile alle variazioni di tensione sulla linea: piccole variazioni, anche quelle che non vengono registrate dagli altri apparecchi elettrici, possono risultare fatali per il sistema che si può bloccare o per le operazioni in corso che daranno risultati senza senso.
Variazioni della tensione di alimentazione possono essere provocate dall'accensione o dallo spegnimento di apparecchi che assorbono molto (lavatrici, lavastoviglie, stufette, ecc.), da interferenze ad alta frequenza, da trasmettitori radio o altri apparecchi elettrici oppure da cadute di tensione sulla linea dovute a cause esterne.
La prima cosa da fare per evitare questo pericolo è di alimentare il sistema con un circuito separato: ciò significa che l'unità e le altre parti del sistema devono essere collegate ad un impianto dipendente da un interruttore generale separato. Potete aggiungere a questo circuito piccoli dispositivi, come una lampadina o un orologio, ma oltre a questi il PC e le sue periferiche devono essere gli unici apparecchi collegati al circuito del sistema.

Se dovete installarlo in casa, però, potrebbe essere difficile riservare un circuito; un'alternativa è quella di non collegare dispositivi che consumano molto alla stessa presa del computer. Gli oggetti da cui dovete guardarvi, soprattutto, sono caffettiere elettriche, lampade, stufette.
Qualsiasi sia l'alimentazione del vostro PC, è di estrema importanza rispettare le prese di terra del sistema. Non dovete infatti escluderle inserendo la spina in una presa con soltanto i due buchi della tensione; le prese di terra, infatti, costituiscono un cammino per scaricare disturbi elettrici e corrente in caso di cortocircuito.
Se avete dei problemi seri con la linea d'alimentazione del computer, potete inserire uno stabilizzatore, che agisca da filtro, tra i componenti del PC e la presa.

## TENERE UN DIARIO DEL SISTEMA

Un "diario di bordo" del sistema è essenziale per il vostro PC; contiene le informazioni che voi registrate ogni volta che modificate il computer o avete dei problemi con questo.
Il diario del sistema dovrebbe essere tenuto in un fascicolo a portata di mano; potreste anche tenerne una copia in un file su disco in modo che sia facile aggiornarlo, ma ricordatevi di avere sempre una copia stampata dell'ultima versione in modo da poter far riferimento a questa quando il sistema non funziona.
Il diario del sistema dovrebbe contenere questo tipo di informazioni:

- La configurazione del sistema: tutti i componenti, comprese le schede inserite nell'unità di sistema (RAM, adattatori, ecc.)
- Informazioni su componenti: produttore di ogni pezzo, hardware o software, insieme a luogo e data dell'acquisto
- Nota degli interventi: sintomi dei vari problemi, come sono stati risolti, data della risoluzione. Questa informazione è utile soprattutto per il tecnico che deve riparare il vostro sistema
- Elenco di numeri telefonici: numero del venditore, del tecnico e altri numeri da chiamare in caso di emergenza.

Tutti quelli che usano il sistema devono conoscere il diario e sapere come usarlo; questo infatti non è utile se non quando impiegato correttamente, e mantenerlo aggiornato implica un piccolo sforzo. Può essere difficile ricordare tutti gli avvenimenti della storia del vostro computer e perciò registrare tutto e subito nel diario può servire in futuro a rendere facile la soluzione di un problema intricato. Questo è il motivo per cui è incalcolabile il valore di un efficiente diario di sistema.

### FARE UNA COPIA DI BACKUP DEL DISCO RIGIDO

Se avete un sistema con un disco rigido (l'XT, per esempio, o un PC con un'unità di espansione) è di vitale importanza che periodicamente facciate una copia di backup dei file contenuti in questo disco così come siete abituati a farla per i file su floppy. Due comandi DOS (BACKUP e RESTORE) vi aiutano in questo compito, permettendovi di selezionare vari metodi di scelta dei file da copiare. Rimandiamo al Capitolo 3 e all'Appendice D per ulteriori informazioni su questi comandi.

### COME MUOVERE IL PC

Il PC-IBM è un sistema ben progettato, costruito per sopportare i problemi di un trasporto, purché vengano prese alcune precauzioni. Si raccomanda vivamente di conservare le confezioni originali in cui è contenuto il sistema al momento dell'acquisto; queste infatti sono particolarmente adatte ad assorbire i colpi e le vibrazioni così consueti durante il trasporto da un posto all'altro.
La parte più delicata dell'IBM è il disco rigido nell'XT o nell'unità di espansione. Se dovete spostare il vostro XT al completo, vi conviene farvi prima una copia di backup di tutti i file importanti che risiedono sul disco rigido, cosa che teoricamente può richiedere fino a 32 dischetti, ma di solito ne occupa molti di meno.
Dopo aver copiato i file del disco rigido, dovete preparare il disco allo spostamento: vi aiuterà il dischetto per la diagnostica del sistema. Inserite il dischetto nel drive A: dell'XT e accendete il sistema: vi apparirà un menu che, tra le altre, presenta l'opzione 3: PREPARE FIXED DISK FOR RELOCATION (prepara il disco rigido per il trasporto). Scegliendo questa opzione, le testine del disco rigido si spostano nella posizione più sicura per un eventuale spostamento dell'XT. Ricordatevi, comunque, che anche questa precauzione non garantisce che i dati presenti sul disco rigido non vadano persi.
Preparare il resto del sistema è un'operazione semplice; inserendo l'apposita protezione (fornita con il sistema) nel drive dei dischi evitate che le testine di lettura si tocchino tra loro. Impacchettate poi il sistema il più attentamente possibile e non avrete problemi.

## 13.2 Come intervenire sul PC

In questo capitolo impareremo cosa fare nel caso in cui qualcosa non funzioni con il PC.

## QUANDO SORGE UN PROBLEMA

Ogniqualvolta incontriate un problema durante una sessione di lavoro, la prima cosa da fare è di riprovare l'ultimo comando che avete utilizzato. Se non accade nulla, provate a far ripartire il programma.
Se siete fermi nel mezzo di un comando DOS, battendo CTRL BREAK o CTRL C fate sospendere l'esecuzione del comando e tornate al prompt del DOS; se invece state eseguendo un programma BASIC quando si verifica il guaio, CTRL BREAK termina l'esecuzione del programma e vi riporta al prompt del BASIC. Se durante una sessione di operazioni BASIC o DOS compare un messaggio d'errore, fate riferimento all'Appendice C o all'Appendice E per maggiori informazioni sulla probabile causa dell'errore.
Se il programma corrente si trova in una condizione tale da non accettare nessuno dei comandi che voi potreste inviare, potete provare a reinizializzare il sistema premendo i tasti CRTL, ALT e DEL.
Osservate che far ripartire il programma o reinizializzare il sistema significa perdere dati creati o elaborati dal programma in corso: questo è uno dei motivi per cui vi conviene abituarvi a fare delle copie dei dischi che contengono dati e programmi importanti.
Se il sistema non risponde nemmeno a questa sollecitazione, togliete tutti i dischi inseriti nei drive, spegnete l'unità di sistema ed attendete per circa 15 secondi; reinserite poi il disco di sistema e quelli con il programma che stavate usando ed accendete il computer: potete ora riprovare l'esecuzione del vostro programma.
Se il problema rimane ancora, probabilmente la causa è da ricercare nell'alimentazione dell'unità centrale, nel disco di sistema o in quelli che contengono il programma o addirittura nell'unità centrale stessa.
Dovete per prima cosa assicurarvi che tutto sia ancora alimentato e acceso. Se sospettate che il difetto sia nella presa potete controllarla inserendo una lampada o qualsiasi altra cosa che sapete essere funzionante. Dovete anche controllare fusibili ed interruttori nei vari apparecchi, ricordandovi, prima di far questo, di togliere il collegamento.
Il passo successivo è di controllare i cavi che uniscono le diverse parti del sistema (ad esempio, il cavo della stampante, del video, della tastiera..). Assicuratevi che siano ben collegati.
Se il vostro problema non è ancora risolto, a questo punto dovete stabilire se questo risiede nell'hardware o nel software della macchina.
Sia che pensiate di risolvere da soli il vostro problema, sia che abbiate deciso di rivolgervi a qualcuno che vi aiuti, è importante riuscire a determinare la natura, hardware o software, del problema. Se volete sistemare da soli il PC, questi sono i primi passi per trovare le cause dei vostri guai.
Queste domande vi aiuteranno a identificare il "colpevole":

— Il problema appare solo con un particolare programma? Se è così, probabilmente la causa sta nel software.
— Il problema sembra dipendere da qualche condizione esterna? Ad esempio, si verifica solo in giornate molto calde o molto secche? In questo caso è più facilmente un problema hardware.
— State utilizzando un nuovo componente o un nuovo programma in un sistema che finora ha sempre funzionato bene? Se così, puntate i vostri sospetti sugli ultimi arrivati.

Le cause dei problemi software si suddividono in due categorie: l'uso di una copia difettosa di un programma o l'uso di un programma difettoso. Se state usando una copia di un programma su disco, come una delle copie di backup del DOS, provate con un'altra copia e ripetete esattamente gli stessi passi che conducevano al problema: se ora il programma funziona correttamente, o la copia che stavate utilizzando era sbagliata (cioè alcune informazioni erano state distrutte da un uso non corretto) oppure avevate commesso un errore nelle operazioni. Se la copia era difettosa, sostituitela e buttatela via così da assicurarvi che non possa più venire utilizzata inavvertitamente.
Se un programma che state usando per la prima volta non funziona, potrebbe anche esserci qualche guaio con l'originale. Se avevate scritto voi il programma, provate a sottoporlo ad un debug; se è un programma BASIC sfruttate le tecniche presentate nel Capitolo 6. Se invece avete comprato il programma, riportatelo al venditore.
I problemi nell'hardware hanno luogo quando qualche pezzo del sistema si rompe: per identificare problemi di questo tipo, dovete usare il programma di diagnostica contenuto nella "Guida Operativa".

## LA GUIDA OPERATIVA

La procedura per la determinazione dei problemi relativa al programma diagnostico compresa nella *Guida Operativa IBM* è progettata per essere utilizzata da chiunque e vi aiuta ad eliminare alcuni dei problemi più comuni con il PC. Lo scopo principale del programma diagnostico è di restringere il campo in cui ricercare le fonti dei problemi identificando l'unità responsabile dell'inconveniente (l'unità di sistema, il video, la tastiera, la stampante); una volta localizzata l'unità difettosa, potete farla controllare da uno specialista.
Il programma è contenuto su disco: quando lo avviate, visualizza un elenco di tutti i dispositivi installati nell'unità di sistema, la quantità di memoria ed ogni interfaccia opzionale. Poi il programma fornisce un menu di opzioni, tra le quali quella di eseguire una serie di programmi di controllo, di formattare o copiare un disco e di uscire al sistema operativo.

Quando scegliete l'opzione di diagnostica, cioè di ricerca degli errori, il PC esegue una serie di controlli che verificano ogni dispositivo presente nell'unità di sistema, compresi schermo, tastiera e stampante. Il programma può anche creare una scheda degli errori trovati.
Se non avete alcuna esperienza tecnica, portate a riparare l'unità difettosa insieme alla scheda degli errori prodotta dal programma scritta su disco. Se invece volete risolvere il problema da soli, vi aiuteranno il manuale IBM *Hardware Maintenance and Service* ed il manuale IBM *Technical Reference*, che sono disponibili presso i rivenditori.

## IL MANUALE HARDWARE MAINTENANCE AND SERVICE

Il manuale IBM *Hardware Maintenance and Service* (HMS) vi fornisce i dati per riparare quasi ogni parte del PC; il manuale è indispensabile per tutti quelli che vogliono riparare da soli il loro computer.
Il manuale HMS fa riferimento alle parti del PC come FRU, o *Field Replaceable Unit* (parti che possono essere sostituite in caso di rottura). Una delle fatiche più gravose nell'uso della letteratura IBM è di non perdere il filo nell'incredibile mondo delle abbreviazioni IBM.
Il manuale HMS fornisce un metodo graduale per identificare la causa di un problema, utilizzando schede di identificazione dei problemi (*Problem Isolation Charts* o PIC). Gli strumenti richiesti sono il programma *Advanced Diagnostic Aids* venduto con il manuale, un comune tester ed uno strumento per rimuovere i circuiti integrati. La sezione PIC è ben organizzata e predisposta in modo che chiunque possieda un minimo di preparazione di base e di esperienza nella riparazione di dispositivi elettronici possa localizzare il problema.
Il programma *Advanced Diagnostic Aids* è in pratica una versione avanzata e più completa del programma diagnostico compreso nella "Guida Operativa". Il capitolo *Diagnostic User's Guide* del manuale HMS propone una serie di strade da percorrere nella ricerca degli errori.
Dopo aver identificato il problema, numerosi paragrafi del manuale HMS vi aiuteranno a scoprire quale sia l'elemento difettoso ed a sostituirlo; il capitolo *Removal/Replacement* comprende procedure ben illustrate che mostrano come accedere all'unità di sistema, ai drive, alla stampante, al video monocromatico, alla tastiera, alle interfacce opzionali. Il paragrafo *Locations* riporta alcuni disegni che mostrano le posizioni dei vari componenti nell'unità di sistema, nella tastiera, nei drive, nella stampante e nelle interfacce opzionali. Infine, quando sia necessario sostituire un pezzo, il *Parts Catalog* nel manuale HMS riporta un elenco esauriente di tutti i pezzi forniti dalla IBM.

## IL MANUALE TECHNICAL REFERENCE

Per chi cerca di risolvere da solo i problemi soprattutto di natura tecnica, il manuale *Technical Reference* è un buon compagno del manuale *Hardware Maintenance and Service*; fornisce infatti molti più dettagli di quest'ultimo. Alcuni brani presuppongono nel lettore una buona conoscenza dell'elettronica e della programmazione in codice macchina.
Il manuale IBM *Technical Reference* è indispensabile per chi voglia progettare circuiti da interfacciare al PC o scrivere programmi in codice macchina. Con questo manuale e gli appropriati data sheets potete farvi una chiara idea di tutto quello che avviene all'interno del PC e delle sue periferiche.

## PROBLEMI COMUNI ED ALCUNE SOLUZIONI

È questo il momento per discutere alcuni dei problemi più comuni e le loro soluzioni. Questi problemi possono essere facilmente identificati una volta che impariate a riconoscerne i sintomi.

### Problemi con il monitor

I sintomi dei problemi con lo schermo sono la mancanza totale di visualizzazione oppure figure confuse, illeggibili. Se non appare nulla sullo schermo e voi siete del tutto sicuri che l'unità di sistema funzioni, controllate l'alimentazione del monitor; controllate il cavo dove si inserisce nella presa (o nel PC, se state usando un video monocromatico). Se è alimentato, assicuratevi che i controlli della luminosità e del contrasto siano disposti in modo che l'immagine sia visibile.
Se tutto quello che avete controllato fin qui è predisposto correttamente, controllate che il cavo tra il PC e il monitor sia inserito in modo sicuro ad entrambe le estremità e che i connettori siano fissati solidamente ai cavi. In particolare, il cavo di terra per i dati dello schermo deve avere un buon contatto elettrico, altrimenti la videata apparirà instabile o confusa.
Se ancora non ottenete la soluzione, il problema probabilmente è localizzato nell'unità di sistema o nella scheda di interfaccia che controlla il monitor (sia un'interfaccia per monitor monocromatico e stampante parallela che una scheda Colore/Grafica). Se non volete curiosare nell'interno dell'unità di sistema, chiamate il rivenditore, altrimenti controllate che la scheda di interfaccia sia collegata saldamente ed esaminate il manuale IBM *Hardware Maintenance and Service* per altre procedure.

**Problemi con la stampante**

Problemi più comuni possono essere un inceppamento o uno scorrimento non corretto della carta, una stampa troppo chiara o errori di software, oppure guasti elettrici o meccanici.
Perché la carta si muova con scorrevolezza, il meccanismo della stampante deve essere sistemato accuratamente e la carta deve entrare direttamente nella stampante. Nella Figura 13.1 potete vedere una stampante con la carta inserita correttamente.

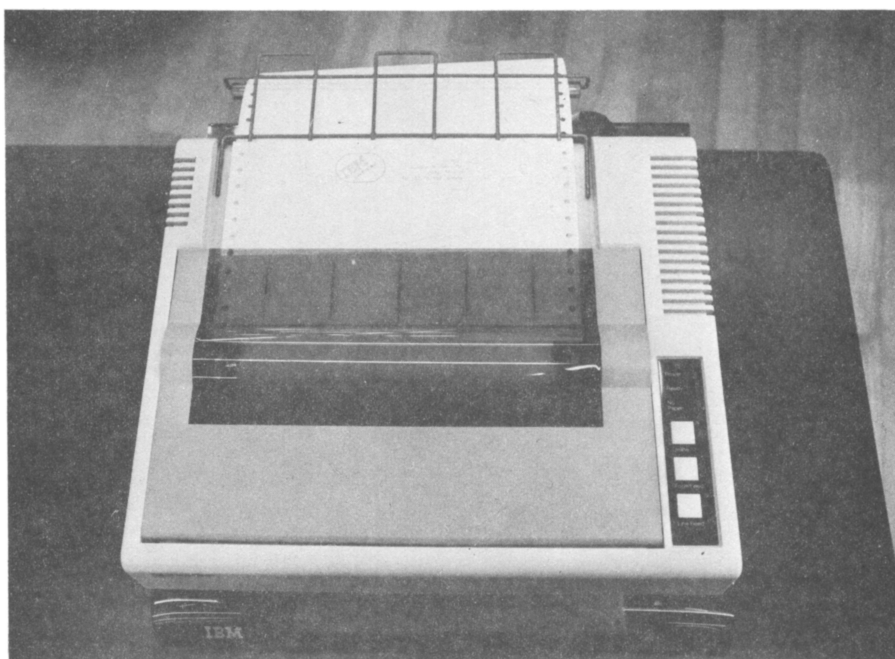



**Figura 13.1** Stampante IBM

Se la stampa è troppo chiara, dovete con molta probabilità sostituire il nastro della stampante seguendo le istruzioni riportate nel manuale della stessa.
Se la stampante non funziona solo con un particolare programma, controllate che il programma sia adatto alla stampante di cui disponete.
Se sospettate un problema elettrico, dapprima controllate i cavi della stampante; i problemi meccanici, come rumori troppo forti o il blocco della testina di stampa, o i problemi nell'elettronica della stampante o nella scheda per l'interfaccia con la stampante nell'unità di sistema, in genere devono essere trattati da personale esperto.

## Problemi con la tastiera

La cosa principale da ricordare a proposito della tastiera è di tenerla pulita. Non versate mai nulla sulla tastiera: è un invito a nozze per i disastri!
Se non ottenete alcuna risposta dalla tastiera, controllate che sia collegata correttamente all'unità di sistema. Nella Figura 13.2 potete vedere il connettore DIN della tastiera.

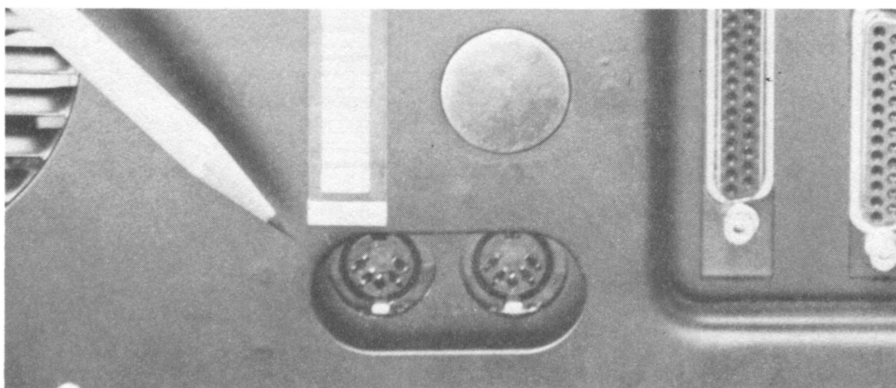


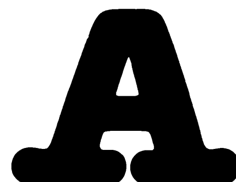
**Figura 13.2** Il connettore DIN della tastiera

## Problemi con i dischi

Quando un dischetto incomincia a procurarvi fastidi, il problema può risiedere nel disco che contiene il programma corrente, in un disco di dati a cui accede il programma o nel drive e nell'elettronica ad esso associata.
Per verificare l'integrità del disco di sistema o dei dati, fate una copia dell'originale e controllatela: se avete dei problemi nello scrivere il disco, assicuratevi che sia stato formattato correttamente e che la tacca di protezione contro la sovrascrittura sia rimossa.
Il problema può essere localizzato nel drive, invece che nel disco e in questo caso, o le testine di lettura sono sporche, o il meccanismo non è allineato perfettamente o la scheda di interfaccia nell'unità di sistema è difettosa. Potete pulire le testine del drive usando dischi particolari che trovate presso i rivenditori. Se il problema sta nel meccanismo del drive o nella sua elettronica, chiamate il rivenditore oppure fate riferimento al manuale IBM *Hardware Maintenance and Service.*

**Appendice**

# Comandi, istruzioni, funzioni e variabili



## A.1 Come utilizzare questa appendice

Lo scopo di questa appendice è quello di descrivere la sintassi di tutte le parole BASIC; per ognuna verranno fornite le seguenti specificazioni:

— **Tipo**
Il tipo è formato da una o più parole tra: istruzione, funzione, comando o variabile; in generale non vi è una differenza assoluta tra questi quattro tipi, ma possono essere differenziati nel seguente modo:

Comando
Solitamente usato in modo diretto, rappresenta un ordine impartito al BASIC affinché compia immediatamente una certa operazione sul programma: ad esempio SAVE (salva il programma nella memoria di massa), RUN (esegue il programma), NEW (cancella la memoria).

Istruzione
Usata normalmente solo all'interno di un programma, esegue un ben determinato processo, quale ad esempio GOTO (salto incondizionato), LET (assegnamento).

Funzione
Per un valore assegnato, restituisce un altro valore che dipende dal tipo di funzione calcolata, ad esempio LOG (calcolo del logaritmo),

COS (calcolo del coseno), HEX$ (conversione in esadecimale); anch'essa viene solitamente usata all'interno di un programma.

Variabile
Contiene un valore generato esternamente al programma, attribuendolo ad un nome convenzionale precedentemente assegnatole, come INKEY$ (valore di input da tastiera), DATE$ (data) e così via.

— **Sintassi**
È la forma nel suo senso più generale; per ulteriori particolari si faccia riferimento a "Convenzioni di nomenclatura e sintassi" a pag. 413.

— **Esempi**
Vengono presentati alcuni degli usi più comuni per ogni termine al fine di illustrare le varie opzioni possibili, senza però la pretesa di esaurire tutte le possibilità.

— **Descrizione**
Con questa parola si indica il testo che segue gli esempi e che ha lo scopo di fornire ulteriori informazioni sulla parola in esame; ogni parametro viene spiegato in dettaglio, e talora vengono esposti degli altri esempi.

## MODO DIRETTO E MODO PROGRAMMATO

Tutte le parole BASIC possono essere usate sia in modo diretto che in modo programmato, a meno che non sia indicato altrimenti.

## LE VERSIONI DEL BASIC

Le versioni del BASIC del PC sono solitamente quattro: Cassette BASIC, Disk BASIC, Advanced BASIC e Compiled BASIC. Sebbene il PC/XT abbia il Cassette BASIC residente su ROM, in pratica non lo potrà mai utilizzare, poiché non possiede alcuna porta per registratore a cassette attraverso la quale salvare e leggere programmi e dati; inoltre, nonostante il modello standard del PC abbia una porta per le cassette, il software necessario per utilizzarla è quasi del tutto inesistente. È per queste ragioni che le differenze tra il Cassette BASIC e le altre versioni non saranno trattate in questa appendice; per ulteriori informazioni consultate il manuale IBM *BASIC*.

## IL TASTO ALT

La maggior parte delle parole BASIC può essere inserita da tastiera per mezzo del tasto ALT, utilizzando un codice di abbreviazione che permette di semplificare la realizzazione dei programmi: ad esempio, premendo i tasti ALT e P contemporaneamente, si ottiene la parola PRINT. Per ognuna delle parole, l'abbreviazione, se esiste, viene indicata tra parentesi subito dopo il titolo.

## CONVENZIONI DI NOMENCLATURA E SINTASSI

Nella presentazione della forma generale di ogni istruzione e di ogni funzione viene usato uno schema standard, le cui convenzioni sono riportate qui di seguito.

| La linea verticale indica che solo uno dei termini deve essere presente. Nell'istruzione reale naturalmente non appare.

[ ] Ogni termine che appare all'interno di parentesi quadre è opzionale. Nell'istruzione reale le parentesi non devono apparire.

... I puntini di sospensione indicano che il termine precedente può essere ripetuto. Anch'essi non devono apparire nell'istruzione reale.

*Numero di linea*
All'inizio di ogni istruzione o linea di istruzioni si sottintende il numero di linea.

*Ulteriore punteggiatura*
Tutti i restanti simboli di punteggiatura, cioè virgola, punto e virgola, virgolette, parentesi, devono apparire come mostrato.

*Maiuscole*
Le parole e le lettere maiuscole devono essere usate esattamente come mostrato.

*Corsivo*
I termini in corsivo hanno un significato generico e non devono essere usati letteralmente; stanno ad indicare che è richiesto un certo tipo di termine. La definizione del termine generale descrive qual è il tipo di termine da usare: quando compare un termine in corsivo, dovrete sostituire una ben precisa espressione o valore in accordo con le definizioni dei termini generali, elencate qui di seguito e nella descrizione delle istruzioni.

## DEFINIZIONI GENERALI

Diamo qui di seguito le abbreviazioni in corsivo più comuni nelle definizioni di istruzioni e funzioni. Ogni altro termine non compreso in questo elenco verrà descritto nel contesto dell'istruzione cui si riferisce.

*indirizzo*
: È una espressione numerica il cui valore è compreso tra 0 e 65535. È sempre relativo al segmento corrente definito dal comando DEF SEG.

*costante*
: È una costante numerica o alfabetica. Le virgolette vengono considerate parte della costante, non come delimitatori.

*n*
: È una costante numerica.

*espressione*
: È una costante, una variabile, una funzione, un'espressione numerica, relazionale o booleana, o una loro qualunque valida combinazione.

*nomefile*
: È ogni possibile nome di file, lungo da uno a 8 caratteri. I caratteri validi sono le lettere dalla A alla Z, i numeri da 0 a 9, e i caratteri ( ) { } @ # $ % ^& ! - ' ' ~ | _ < > \. Il primo carattere deve essere una lettera.

*specfile*
: È un nome valido di file preceduto opzionalmente dal nome di un dispositivo e/o dal cammino in un directory. Nomi validi per dispositivi sono:

| | | | |
|---|---|---|---|
| KYBD: | SCRN: | LPT1: | LPT2: |
| LPT3: | COM1: | COM2: | CAS1: |
| A: | B: | C: | D: |

*linea*
: È una costante intera che corrisponde ad un numero di linea esistente.

*esprnum*
: È una costante o variabile numerica, oppure una funzione, o una qualsiasi loro valida combinazione.

*varnum*
È un nome di variabile numerica o di un elemento di array.

*varstr*
È un nome di variabile stringa, sottostringhe comprese.

*stringa*
È una costante o una variabile stringa, o una sottostringa, oppure una funzione che restituisce una stringa.

*variabile*
È un nome di variabile numerica o di tipo stringa, sottostringhe e vettori esclusi.

# A.2 Elenco

## ABS Funzione

Restituisce il valore assoluto dell'espressione *esprnum*.

*Sintassi:*
*varnum* = ABS(*esprnum*)

*Esempio*:

```
Y=ABS(X)
```

Il valore assoluto di un numero è sempre positivo o uguale a zero.

## ASC Funzione

Restituisce il valore in codice ASCII per il primo carattere di *stringa*.

*Sintassi:*
*varnum* = ASC("*stringa*")

*Esempio:*

```
PRINT ASC("CIAO")
```

Il risultato della funzione ASC è il valore numerico in codice ASCII del primo carattere di *stringa*; se la stringa è vuota, il sistema restituisce il messaggio d'errore ILLEGAL FUNCTION CALL.
Esattamente l'inverso della ASC è la funzione CHR$, che converte da codice ASCII a carattere.

## ATN Funzione

Restituisce l'arcotangente di *esprnum*.

*Sintassi:*
*varnum* = ATN(*esprnum*)

*Esempio:*

```
PRINT ATN(3)
```

La funzione ATN restituisce il valore dell'angolo la cui tangente è *esprnum*; il risultato è espresso in radianti nell'intervallo da $-\pi/2$ a $+\pi/2$, dove $\pi = 3.141593$. Per convertire da radianti a gradi è sufficiente moltiplicare per $180/\pi$.
La funzione ATN opera in precisione semplice a meno che il BASIC non venga avviato, con l'opzione /D, in doppia precisione.

## **AUTO** (ALT A) Comando

Genera automaticamente il numero di linea successivo ogniqualvolta viene premuto il tasto ENTER. Non è utilizzabile nel Compiled BASIC.

*Sintassi:*
AUTO [*linea*] [,[*incremento*]]

*Esempi:*

```
AUTO
AUTO 15,5
AUTO .,
```

Il parametro *linea* è il numero di linea da cui partire per la numerazione automatica, e può essere sostituito con un punto per indicare che la numerazione deve iniziare e partire dalla linea corrente; *incremento* è il valore che deve essere aggiunto ad ogni numero di linea per ottenere il successivo.
La numerazione inizia dalla linea *linea* ed incrementa ogni successivo numero di linea del valore *incremento*. Quando vengono omessi entrambi i valori, vengono posti come valori di default 10,10. Se non si specifica il valore di *incremento* sostituendolo con una virgola, viene utilizzato il valore di *incremento* presente nell'ultimo comando AUTO. Infine, omettendo solo *linea*, la numerazione inizierà dalla linea 0.
Il comando AUTO viene solitamente usato nella battitura dei programmi e vi risparmia il compito di scrivere ogni volta il numero di linea.
Nel caso in cui AUTO generi un numero di linea già esistente nel programma, subito dopo il numero comparirà un asterisco per avvisarvi che l'introduzione di quella nuova linea porterebbe alla cancellazione di quella già presente. Comunque, se immediatamente dopo la comparsa dell'asterisco premete il tasto ENTER, la linea già presente non viene rimpiazzata ed il comando AUTO genera un altro numero.
Il comando AUTO termina di operare quando premete il tasto CTRL BREAK; la linea alla quale questa operazione è compiuta non viene salvata ed il BASIC ritorna al livello comandi.

NOTA: quando state operando nel modo AUTO, potete fare cambiamenti solo alla linea corrente; se volete modificare un'altra linea dello schermo, dovete uscire dal modo AUTO premendo CTRL BREAK.

## **BEEP** Istruzione

L'altoparlante emette un beep.

*Sintassi:*
BEEP

*Esempio:*

```
IF X<20 THEN BEEP
```

L'istruzione BEEP fa suonare l'altoparlante ad una frequenza di 800 Hz per 0.25 secondi, ed ha lo stesso effetto di:

PRINT CHR$(7)

## **BLOAD** Comando

Carica in memoria un file binario.

*Sintassi:*
BLOAD "*specfile*" [,*offset*]

*Esempi:*

```
BLOAD"QUADRO",0
BLOAD"A:MANIPOLA"
```

*offset* è un numero compreso tra 0 e 65535 che rappresenta l'indirizzo a partire dal quale inizia il caricamento. È definito in relazione al segmento dichiarato dall'ultima istruzione DEF SEG. Se viene omesso, come valore di *offset* viene utilizzato quello del comando BSAVE, cioè il file viene caricato a partire dalla stessa locazione dalla quale era stato salvato.
Quando il comando BLOAD va in esecuzione, il file viene caricato in memoria a partire dalla specifica locazione; quando è omesso il nome del dispositivo di memoria, il DOS utilizza il drive di default.
I comandi BLOAD e BSAVE sono usati per caricare e salvare programmi in linguaggio macchina (potete usare un programma in linguaggio mac-

china dall'interno di un programma BASIC richiamandolo con l'istruzione CALL); comunque questa istruzione non è ristretta ai soli programmi in linguaggio macchina. Ogni segmento può essere utilizzato sia come destinazione che come sorgente per questa istruzione grazie all'istruzione DEF SEG. Per esempio voi avete a disposizione un mezzo potente per salvare e visualizzare immagini salvando e caricando il buffer dello schermo. Attenzione: il BASIC non compie alcun controllo sull'indirizzo che viene passato; è possibile cioè caricare ovunque nella memoria, anche in quelle parti in cui non bisogna assolutamente caricare programmi-utente, quali l'area dello stack o l'area delle variabili, o l'area in cui sono già presenti altri programmi BASIC.

## **BSAVE** (ALT B) Comando

Salva una parte della memoria del computer in un dispositivo specifico.

*Sintassi:*

BSAVE "*specfile*", *offset*, *lunghezza*

*Esempi:*

```
BSAVE"QUADRO",0,&H4000
BSAVE"A:MANIPOLA",0,100
```

Il parametro *offset* è un numero compreso tra 0 e 65535 che rappresenta l'indirizzo da cui salvare il programma, relativamente al segmento dichiarato nell'ultima istruzione DEF SEG. *lunghezza* è un numero compreso tra 1 e 65535 che indica la lunghezza dell'immagine di memoria da salvare. Se vengono omesse sia *lunghezza* che *offset*, il sistema darà SYNTAX ERROR e il comando non verrà eseguito. Se invece viene omesso il nome del dispositivo in cui salvare il file, viene usato il drive di default.
I comandi BLOAD e BSAVE sono usati per caricare e salvare programmi in linguaggio macchina (potete usare un programma in linguaggio macchina dall'interno di un programma BASIC richiamandolo con l'istruzione CALL); comunque questa istruzione non è ristretta ai soli programmi in linguaggio macchina. Ogni segmento può essere utilizzato sia come destinazione che come sorgente per questa istruzione grazie all'istruzione DEF SEG. Per esempio, voi avete a disposizione un potente mezzo per salvare le immagini che appaiono sullo schermo per mezzo del buffer dello schermo, come mostrato nel primo dei due esempi precedenti, facendolo precedere dal comando

DEF SEG=&HB800

## **CALL** Istruzione

Chiamata ad una subroutine in linguaggio macchina.

*Sintassi:*
CALL*varnum*[(*variabile*[,*variabile*]...)]

*Esempi:*

```
CALL Z(A,B$,C)
CALL M
```

Il valore di *varnum* indica l'indirizzo di memoria di inizio della subroutine chiamata come un offset nel corrente segmento di memoria (come definito dall'ultima istruzione DEF SEG). *variabile* è il nome che deve essere passato come argomento alla subroutine.
L'istruzione CALL è un metodo di interfacciamento tra programmi in linguaggio macchina e programmi in BASIC; un ulteriore metodo è offerto dalla funzione USR.

## **CDBL** Funzione

Converte *esprnum* in un numero in doppia precisione.

*Sintassi:*
*varnum* = CDBL(*esprnum*)

*Esempio:*

```
PRINT CDBL(S)
```

*esprnum* può essere una qualsiasi espressione numerica.

## **CHAIN** Istruzione

Trasferisce controllo e variabili dal programma corrente ad un altro programma.

*Sintassi:*
CHAIN [MERGE]"*specfile*" [,[*linea*] [,[ALL] [,DELETE *intervallo*]]]

*Esempi:*

```
CHAIN "A:PROG1"
CHAIN "B:INTRO",130
CHAIN MERGE "A:OVRLAY"
CHAIN "A:TEST",,ALL
CHAIN MERGE "A:PART2",1000, DELETE 1000-5000
```

Il nome del file contenuto in *specfile* è quello del file al quale si desidera passare il controllo. *linea* è un numero di linea, oppure un'espressione che calcola il numero di una linea all'interno del programma da concatenare: questo numero specifica la linea a partire dalla quale il programma concatenato dovrà essere eseguito. Se *linea* viene omesso, l'esecuzione inizia dalla prima linea del programma concatenato.
Inoltre *linea* non è interessata da un comando RENUM; infatti se si procede alla rinumerazione di un programma concatenato, tutte le istruzioni CHAIN dovranno essere cambiate in modo che puntino al nuovo numero di linea.
L'opzione ALL specifica che ogni variabile del programma corrente sarà passata al programma concatenato; se non viene usata, dovrete inserire nel programma concatenante un'istruzione COMMON per passare le variabili al programma concatenato.
MERGE tratta una parte di codice nel programma BASIC come se fosse un *overlay* (overlay significa sovrapposizione ed è una tecnica che permette di utilizzare in tempi diversi la stessa area di memoria principale caricandovi, durante l'esecuzione, differenti porzioni di programmi contenuti in memorie secondarie); vien fatta cioè un'operazione di MERGE tra il programma concatenante ed il programma concatenato. Per poter essere "fuso" il programma concatenato deve essere un file in codice ASCII.
Dopo aver compiuto un'operazione di overlay, si pone il problema di cancellarla in modo di poterne ripetere eventualmente un'altra: per far ciò basta usare l'opzione DELETE, che di fatto agisce come il comando DELETE. Nell'ultimo esempio vengono cancellate le linee dalla 1000 alla 5000 del programma concatenante prima di caricare in overlay il programma concatenato. I numeri di linea in *intervallo* sono interessati dal comando RENUM.

NOTE:

1. L'istruzione CHAIN lascia aperti i file.
2. L'istruzione CHAIN con l'opzione MERGE preserva la corrente OPTION BASE.
3. Se viene omessa l'opzione MERGE, l'OPTION BASE non viene conservata nel programma concatenato. Inoltre, senza MERGE,

l'istruzione CHAIN non conserva i tipi di variabile o le funzioni definite dall'utente per utilizzarle nel programma immerso. Ciò significa che ogni istruzione tipo DEFINT, DEFSNG, DEFDBL, DEF FN contenente delle variabili condivise deve essere ripetuta nel programma concatenato.

## **CHDIR** Comando

Permette di cambiare il catalogo corrente, ed esiste solo in Advanced BASIC e Disk BASIC.

*Sintassi:*
CHDIR"*cammino*"

*Esempi:*

```
CHDIR "\STUDENTI\VOTI"
CHDIR "\"
CHDIR ".."
```

*cammino* è una stringa che identifica il catalogo che diventerà il nuovo catalogo corrente; per ulteriori informazioni sulla struttura ad albero si faccia riferimento al Capitolo 3. Inoltre *cammino* non deve superare in lunghezza i 63 caratteri. Per una discussione su come creare e cancellare cataloghi si vedano i comandi MKDIR e RMDIR.

## **CHR$** Funzione

Converte un codice ASCII nel corrispondente carattere.

*Sintassi:*
*varstr* = CHR$(*esprnum*)

*Esempi:*

```
PRINT CHR$(237)
IF X$=CHR$(8) THEN 100
```

Il termine *esprnum* deve essere compreso tra 0 e 255. La funzione CHR$ restituisce il singolo carattere il cui codice ASCII è *esprnum*, ed è comunemente usata per inviare caratteri speciali allo schermo o alla stampante; per esempio il carattere BEL, che fa produrre al PC il caratteristico

suono beep, può essere inserito in un programma come CHR$(7) prima di un messaggio d'errore; ovviamente può essere usata anche l'istruzione BEEP. Per l'operazione inversa si veda la funzione ASC.

## **CINT** Funzione

Converte *esprnum* in un intero.

*Sintassi:*
*varnum* = CINT(*esprnum*)

*Esempi:*

```
PRINT CINT(45.67)
Q%=CINT(X*3)
```

Se *esprnum* non è contenuta nell'intervallo −32768,+32768, verrà visualizzato un messaggio d'errore di overflow. *esprnum* viene convertita in un intero arrotondando la parte frazionaria. Vedere anche le funzioni FIX e INT: entrambe restituiscono degli interi. Vedere inoltre le funzioni CDBL e CSNG per convertire numeri in semplice o doppia precisione.

## **CIRCLE** Istruzione

Disegna sullo schermo un'ellisse con centro nel punto di coordinate (*x*,*y*) e raggio *r*. Utilizzabile solo in Advanced BASIC e Compiled BASIC, in modo Grafico.

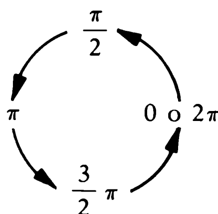
*Sintassi:*
CIRCLE(*x*,*y*), *r*[,*colore*[,*inizio*,*fine*[,*rapporto*]]]

*Esempi:*

```
CIRCLE (150,100),100,2
CIRCLE (160,250),200,2,-(2*PI/3),-(PI/3)
CIRCLE (160,100),50,,,,1
```

Le coordinate del centro dell'ellisse sono date da (*x*,*y*) e possono essere passate sia in forma assoluta che in forma relativa. Il raggio dell'ellisse, o asse maggiore, è dato da *r*. *colore* è un numero compreso tra 0 e 3 che specifica quale deve essere il colore dell'ellisse; in media risoluzione *colore* seleziona il colore dalla corrente tavolozza di colori definita con

l'istruzione COLOR; lo zero rappresenta il colore di fondo. Il valore di default è il colore del testo, il numero 3. In alta risoluzione 0 significa nero, ed il valore di default 1 rappresenta il bianco. I termini *inizio* e *fine* sono gli angoli, in radianti, con valori compresi tra $-2\pi$ e $+2\pi$, dove $\pi=3.141593$, che specificano dove deve iniziare e terminare il disegno dell'ellisse; gli angoli sono posizionati nel modo matematico standard, con lo 0 a destra e l'incremento in senso antiorario.




Se *inizio* e *fine* hanno valori negativi (si noti che $-0$ non è concesso), l'ellisse sarà connessa al punto di centro con una linea e gli angoli saranno trattati come se fossero positivi (si osservi che non è la stessa cosa che sommare $2\pi$). L'angolo di inizio deve essere maggiore o minore dell'angolo di fine; ad esempio:

```
10 pi=3.141593
20 SCREEN 1
30 CIRCLE (160,100),60,,-pi,-pi/2
```

disegnerà una parte di circonferenza.
*rapporto* è una espressione numerica che riguarda il rapporto tra il raggio *x* ed il raggio *y*; essa ha valore di default 5/6 in media risoluzione e 5/12 in alta risoluzione. Questi valori fanno sì che l'ellisse sia un cerchio, dato che il rapporto normale tra le dimensioni dello schermo è di 4/3.
Se *rapporto* ha un valore minore di 1, allora *r* ha il valore di *x*; cioè il raggio è misurato in punti in direzione orizzontale. Se invece *rapporto* è maggiore di 1, *r* ha il valore di *y*. Per esempio:

```
10 SCREEN 1
20 CIRCLE (160,100),60,,,,5/18
```

disegnerà un'ellisse.
In parecchi casi, il valore 1 per *rapporto* darà circonferenze ben fatte in media risoluzione; inoltre, ciò farà sì che la circonferenza venga disegnata piuttosto rapidamente. L'ultimo punto di riferimento, dopo che è stata disegnata la circonferenza, è il centro della circonferenza stessa. I punti che risultanó al di fuori dello schermo non vengono disegnati dall'istruzione CIRCLE.

## CLEAR Comando

Assegna il valore zero a tutte le variabili numeriche e cancella il contenuto di tutte le variabili stringa. Un'opzione permette di definire l'area di memoria su cui operare e la dimensione dello spazio riservato allo stack.

*Sintassi:*

CLEAR [,[*byte*] [,*stack*]]

*Esempi:*

```
CLEAR
CLEAR , 32768
CLEAR ,, 2000
CLEAR , 32768, 2000
```

Il termine *byte* è un numero che, se specificato, indica le dimensioni massime in byte per l'area di lavoro del BASIC (dove i vostri programmi e dati sono immagazzinati, oltre all'area di lavoro dell'interprete); di fatto probabilmente userete *byte* quando dovrete riservare dello spazio in memoria per programmi in linguaggio macchina. L'opzione *stack* riserva una parte dello stack per il BASIC, ed il suo valore di default è 512 byte o comunque un ottavo della memoria disponibile, qualora le dimensioni di quest'ultima fossero inferiori. Vi sarà utile inserire l'opzione *stack* se il vostro programma usa un gran numero di istruzioni GOSUB o di loop FOR-NEXT annidati, o un'istruzione PAINT per creare delle immagini complesse.
Il comando CLEAR libera tutta la memoria usata per i dati senza peraltro cancellare il programma corrente; dopo aver inviato il comando CLEAR, gli array restano indefiniti, le variabili numeriche vengono poste uguali a zero, le variabili stringa hanno contenuto nullo e tutte le informazioni definite con una qualsiasi istruzione DEF vengono perdute, comprese DEF FN, DEF SEG, DEF USR, DEFINT, DEFDBL, DEFSNG, DEFSTR.
Con l'esecuzione di questo comando viene fermata la musica in esecuzione in quel momento ed annullato il sottofondo musicale; inoltre PEN e STRIG sono posti nella condizione di disattivazione. L'istruzione ERASE può invece essere usata per liberare parte della memoria senza per questo cancellare tutti i dati del programma; infatti cancella dall'area di lavoro solo gli array specificati.

## **CLOSE** Istruzione

Termina le operazioni di input/output con un dispositivo o un file.

*Sintassi:*
CLOSE [[#]*numfile*[,[#]*numfile*]...]

*Esempio:*

```
CLOSE #1,#2
```

Il parametro *numfile* è il numero che è stato usato nell'istruzione OPEN; l'associazione tra un dispositivo od un particolare file ed il suo numero di file cessa non appena viene eseguita CLOSE. Non sarà valida una operazione successiva di input/output che specifica quel numero di file. Il file o il dispositivo può essere nuovamente aperto usando lo stesso, oppure un differente numero di file, o il numero di file può essere riusato per aprire un qualsiasi file o dispositivo.
Un'istruzione CLOSE per un file o un dispositivo aperto per output sequenziali provoca la scrittura del buffer finale nel file o nel dispositivo stesso. Invece un'istruzione CLOSE senza la specificazione *numfile* provoca la chiusura di tutti i file e di tutti i dispositivi precedentemente aperti.
Si noti che anche l'esecuzione di END, NEW, RESET, RUN senza l'opzione R causa automaticamente la chiusura di tutti i file e dispositivi aperti, cosa che non avviene con STOP. (Vedere OPEN per ulteriori informazioni sull'apertura dei file).

## **CLS** Istruzione

Cancella lo schermo.

*Sintassi:*
CLS

*Esempio:*

```
CLS
```

Se lo schermo si trova nel modo Text, la pagina attiva (vedi SCREEN) viene cancellata con il colore di fondo (vedi COLOR). Se lo schermo si trova nel modo Grafico, in media o in alta risoluzione, l'intero buffer di schermo viene cancellato sempre con il colore del fondo.

L'istruzione CLS riporta inoltre il cursore nella posizione home: in modo Text significa che il cursore ricompare nella posizione in alto a sinistra dello schermo; in modo Grafico, invece, significa che l'ultimo punto di riferimento, per ogni futura istruzione grafica è il punto di centro dello schermo, cioè (160,100) in bassa risoluzione, e (320,100) in alta risoluzione.
Cambiando il modo dello schermo o la sua larghezza, rispettivamente con l'istruzione SCREEN o WIDTH, lo schermo viene anche automaticamente cancellato. Infine, anche premendo i tasti CTRL HOME si ottiene la cancellazione dello schermo.

NOTA: dopo che è stata usata l'istruzione VIEW, CLS cancella solamente la finestra corrente; per cancellare l'intero schermo dovrete disabilitare la finestra e quindi usare CLS. Per ulteriori informazioni sulle finestre vedere VIEW.

## **COLOR** (ALT C) Istruzione

Cambia il colore del primo piano, del fondo e del bordo dello schermo; in modo Text si possono fare le seguenti scelte:

— 16 colori per il testo
— carattere lampeggiante
— 8 colori per il fondo
— 16 colori per il bordo

in modo Grafico invece:

— 16 colori per il fondo
— 2 tavolozze di 3 colori ciascuna
— ha lo stesso colore del fondo

### In modo Text

*Sintassi:*

COLOR [*testo*] [,[*fondo*] [,*bordo*]]

*Esempi:*

```
COLOR 14,1,0
COLOR ,5
```

Il termine *testo* è un numero compreso tra 0 e 31 che rappresenta il colore dei caratteri; *fondo* invece è un numero dell'intervallo da 0 a 31 per il

colore del fondo; infine *bordo* tra 0 e 15, è il numero del colore del bordo dello schermo.
Se possedete la scheda Colore/Grafica, per il *testo* potrete utilizzare i seguenti colori:

| | | | | | |
|---|---|---|---|---|---|
| 0 | Nero | 6 | Marrone | 12 | Rosso chiaro |
| 1 | Blu | 7 | Bianco | 13 | Magenta chiaro |
| 2 | Verde | 8 | Grigio | 14 | Giallo |
| 3 | Cyan | 9 | Blu chiaro | 15 | Bianco brillante |
| 4 | Rosso | 10 | Verde chiaro | | |
| 5 | Magenta | 11 | Cyan chiaro | | |

I colori e la loro intensità possono dipendere molto dal vostro schermo; può semplificare pensare ai valori da 8 a 15 come a quelli relativi ai colori chiari, oppure ai valori da 0 a 7 per i colori ad alta intensità.
Potete ottenere il carattere lampeggiante ponendo *testo* uguale a 16 più il valore del colore desiderato; un valore da 16 a 31 produce quindi un carattere lampeggiante. Per il *fondo* invece potete selezionare solo colori da 0 a 7. Se disponete per il vostro PC dell'interfaccia per video monocromatico e stampante parallela, per *testo* potete usare i seguenti valori:

| | |
|---|---|
| 0 | Nero |
| 1 | Carattere sottolineato bianco |
| 2-7 | Bianco |

Analogamente a quanto visto per il caso della scheda Colore/Grafica, aggiungendo 8 al valore del colore desiderato si ottiene il colore medesimo ma ad alta intensità; per esempio, il valore 15 è relativo al bianco ad alta intensità. Il valore 9, invece, vi dà il bianco ad alta intensità, ma sottolineato. L'unico colore per il quale non è possibile ottenere l'alta intensità è il nero.
Esattamente come con la scheda Colore/Grafica, aggiungendo 16 al numero del colore desiderato si ottiene il lampeggio del carattere; così 16 vi darà il nero lampeggiante, e col numero 31 potrete ottenere i caratteri in bianco ad alta intensità lampeggianti.
Per il colore di fondo, con l'interfaccia per video monocromatico e stampante parallela, si possono selezionare i seguenti colori:

| | |
|---|---|
| 0-6 | Nero |
| 7 | Bianco |

NOTA: il bianco (colore 7) come colore di fondo ci appare come bianco, su di uno schermo monocromatico IBM, solo quando viene usato

con un colore di testo di valore 0, 8, 16, 24, cioè nero. Questo genera dei caratteri diversi.
Il nero (valori 0, 8, 16, 24) a sua volta ci appare come nero per il testo solo se il colore di fondo è il bianco, con valore 0, creando dei caratteri in reverse.
Altre combinazioni di colori per fondo e testo producono sullo schermo monocromatico dei risultati standard, ovvero bianco e nero.
Note per ulteriori adattamenti:

1. Il colore del testo può essere uguale al colore del fondo, però così facendo ogni carattere visualizzato sullo schermo risulta invisibile. Un semplice cambiamento del colore del fondo o del testo farà riapparire i caratteri.
2. Ogni parametro può essere omesso, e così facendo il PC assumerà il vecchio valore.
3. Se l'istruzione COLOR termina con una virgola, il colore cambierà, ma provocherete la visualizzazione del messaggio d'errore MISSING OPERAND. Ad esempio:
   COLOR ,7,
   non è un'istruzione valida.
4. Ogni valore assegnato ai parametri al di fuori dell'intervallo da 0 a 255, provoca la visualizzazione del messaggio d'errore ILLEGAL FUNCTION CALL. In questo caso vengono mantenuti validi i valori precedenti.

## **In modo Grafico** (solo in media risoluzione)

*Sintassi:*

COLOR [*fondo*] [,*tavolozza*]]

*Esempi:*

```
COLOR ,1
COLOR 7
```

Il parametro *fondo* è un numero che specifica il colore del fondo dello schermo; i colori permessi ed i loro relativi valori sono gli stessi di quelli trattati precedentemente. Invece *tavolozza* è un numero che seleziona la tavolozza dei colori.
I colori selezionati quando scegliete una tavolozza sono i seguenti:

| Colore | Tavolozza 0 | Tavolozza 1 |
|---|---|---|
| 1 | Verde | Cyan |
| 2 | Rosso | Magenta |
| 3 | Marrone | Bianco |

Se *tavolozza* è un numero pari, viene selezionata la tavolozza 0, che associa i colori verde, rosso e marrone ai numeri 1, 2 e 3. La tavolozza 1 viene selezionata ogni volta che a *tavolozza* si sostituisce un valore dispari. Il colore selezionato per *fondo* può essere lo stesso di ogni colore della tavolozza.
Anche in questo caso ogni parametro può essere omesso, ed ancora vengono assunti i vecchi valori.
In modo Grafico l'istruzione COLOR stabilisce il colore del fondo ed una tavolozza di tre colori; potrete selezionare ciascuno di questi quattro colori con le istruzioni PSET, PRESET, LINE, CIRCLE, PAINT e DRAW.
L'istruzione COLOR ha significato solo in media risoluzione (introducibile per mezzo dell'istruzione SCREEN 1); usandola in alta risoluzione si otterrà un messaggio d'errore ILLEGAL FUNCTION CALL. Questo stesso messaggio verrà visualizzato nel caso in cui passaste nell'istruzione un valore al di fuori dell'intervallo da 0 a 255, nel qual caso verrebbe mantenuto valido il valore precedente.

## **COM (n)** Istruzione

Abilita o disabilita l'intercettamento di attività di comunicazione ad una specifica interfaccia di comunicazione. Utilizzabile solo in Advanced BASIC e in Compiled BASIC.

*Sintassi:*

COM(*esprnum*)ON
COM(*esprnum*)OFF
COM(*esprnum*)STOP

*Esempi:*

```
COM (1) ON
COM (R) OFF
COM (2) STOP
```

Il parametro *esprnum* è il numero dell'interfaccia di comunicazione (1 o 2).
L'istruzione COM(*esprnum*)ON deve essere eseguita per permettere l'intercettamento dell'istruzione ON COM(*esprnum*). Dopo COM(*esprnum*)ON, se nell'istruzione ON COM(*esprnum*) viene specificato un numero di linea diverso da zero, il BASIC controlla se un carattere è stato introdotto nell'interfaccia di comunicazione ogni volta che viene eseguita una nuova istruzione.
Se invece l'istruzione è COM(*esprnum*)OFF, non si verifica alcun intercet-

tamento e non viene ricordata alcuna attività di comunicazione. Se infine è stata eseguita un'istruzione COM(*esprnum*)STOP, non avviene alcun intercettamento, ma ogni attività di comunicazione viene ricordata in modo che non appena sia eseguita un'istruzione COM(*esprnum*)ON si verifichi un intercettamento.

## **COMMON** Istruzione

Passa le variabili ad un programma concatenato.

*Sintassi:*
COMMON *variabile*[,*variabile*]...

*Esempio:*

```
COMMON A,C,D()
```

Il parametro *variabile* è un nome di variabile che viene passato al programma concatenato; le variabili di tipo array vengono specificate aggiungendo i simboli ( ) al nome della variabile.
L'istruzione COMMON, che viene usata unitamente all'istruzione CHAIN, può apparire ovunque in un programma, nonostante sia raccomandabile inserirla all'inizio; inoltre, all'interno di un programma può essere inserito un numero qualsiasi di istruzioni COMMON, ma una variabile può apparire al più in una di esse. Nel caso in cui si desiderasse passare tutte le variabili, è possibile usare l'istruzione CHAIN con l'opzione ALL, omettendo l'istruzione COMMON.
Gli array che vengono passati non devono essere dimensionati nel programma concatenato.

## **CONT** Comando

Riprende l'esecuzione di un programma dopo un'interruzione. Non è utilizzabile in Compiled BASIC.

*Sintassi:*
CONT

*Esempio:*

```
CONT
```

Il comando CONT può essere usato per riprendere l'esecuzione di un pro-

gramma dopo la pressione dei tasti CTRL BREAK, o dopo l'esecuzione di una istruzione STOP o END, o anche dopo che il programma è incorso in un errore; l'esecuzione riprende esattamente dal punto in cui è avvenuta l'interruzione. Se poi l'interruzione avviene dopo l'apparizione del prompt di un'istruzione di INPUT, l'esecuzione continua con la ripresentazione del prompt.
Il comando CONT è solitamente usato unitamente all'istruzione STOP in fase di debugging (processo di localizzazione e rimozione degli errori e dei malfunzionamenti in un programma). Quando l'esecuzione viene fermata, potete esaminare o cambiare i valori delle variabili usando le istruzioni in modo diretto, facendo poi ripartire l'esecuzione con il comando CONT. Potete anche usare in modo diretto GOTO, che riprende l'esecuzione a partire da un particolare numero di linea.
CONT non è utilizzabile se il programma è stato creato durante l'interruzione.

## **COS** Funzione

Restituisce il coseno di *angolo*.

*Sintassi:*
*varnum* = COS(*angolo*)

*Esempio:*

```
PRINT COS(3.141593)
```

L'angolo di cui si desidera calcolare il coseno è rappresentato da *angolo*, valore che deve essere espresso in radianti. Per convertire un angolo da gradi a radianti basta moltiplicare il valore in gradi per $\pi/180$, dove $\pi = 3.141593$. Il calcolo di COS(*angolo*) viene normalmente eseguito in precisione semplice, oppure in doppia precisione quando il BASIC viene avviato con l'opzione /D.

## **CSNG** Funzione

Converte l'argomento in un numero in precisione semplice.

*Sintassi:*
*varnum* = CSNG(*esprnum*)

*Esempio:*

```
PRINT CSNG(975.3421222#)
```

## **CSRLIN** Variabile

Contiene la coordinata verticale del cursore.

*Sintassi:*
*varnum* = CSRLIN

*Esempio:*

```
PRINT CSRLIN
```

La variabile CSRLIN contiene il numero della riga della posizione corrente del cursore nella pagina attiva (il concetto di pagina attiva è spiegato in dettaglio nell'istruzione SCREEN). Il valore restituito apparterrà all'intervallo da 0 a 25. La funzione POS restituisce invece la colonna in cui si trova il cursore (vedi POS). Per informazioni su come posizionare il cursore vedere LOCATE.

## **CVI, CVS, CVD** Funzioni

Convertono una variabile di tipo stringa in una variabile di tipo numerico.

*Sintassi:*
*varnum* = CVI(*stringa di 2 byte*)
*varnum* = CVS(*stringa di 4 byte*)
*varnum* = CVD(*stringa di 8 byte*)

*Esempi:*

```
F%=CVI(N$)
M!=CVS(B$)
X#=CVD(Z$)
```

Il valore numerico che viene letto da un file ad accesso diretto può essere convertito da stringa a numero: CVI converte una stringa di 2 byte in un numero intero, CVS converte una stringa di 4 byte in un numero in precisione semplice, CVD converte una stringa di 8 byte in un numero in doppia precisione.
Le funzioni CVI, CVS, CVD non possono cambiare i byte dei dati attuali, cambiano solamente il modo in cui il BASIC interpreta questi dati (vedi MKI$, MKS$, MKD$).

## DATA Istruzione

Contiene le costanti numeriche e di tipo stringa, utilizzate dalle istruzioni READ.

*Sintassi:*
DATA *costante*[,*costante*]...

*Esempio:*

```
DATA 45,34,EDIGEO
```

Il parametro *costante* deve essere una costante numerica o di tipo stringa; nella lista non sono consentite espressioni. La costante numerica può apparire in un qualsiasi formato — intera, virgola fissa, virgola mobile, esadecimale, ottale. Le costanti di tipo stringa nelle istruzioni DATA non devono essere racchiuse tra virgolette, a meno che la stringa non contenga virgole, punti e virgola, o spazi bianchi significativi all'inizio o alla fine. Le istruzioni DATA non sono eseguibili e se ne possono inserire tante quante si desidera in un punto qualsiasi del programma; inoltre ognuna di esse può contenere tante costanti quante ce ne stanno in una linea. Il contenuto dell'istruzione DATA può essere pensato come una lista continua di termini, senza tenere conto di quanti termini stiano in una linea, o dove le linee siano poste all'interno del programma. L'istruzione READ accede alle istruzioni DATA secondo l'ordine dei numeri di linea.
Il tipo di variabile, numerica o stringa, contenuto nell'istruzione READ deve essere in accordo con la corrispondente costante nell'istruzione DATA, altrimenti si incorrerà in un SYNTAX ERROR.
Per rileggere informazioni da ogni linea della lista delle istruzioni DATA si può usare l'istruzione RESTORE.

## DATE$ Istruzione

Inizializza la data.

*Sintassi:*
DATE$ = "*stringa*"

*Esempio:*

```
DATE$="03-15-1985"
DATE$=D$
```

Potete introdurre il parametro *stringa* in uno qualsiasi dei seguenti modi:

*mm-gg-aa*
*mm/gg/aa*
*mm-gg-aaaa*
*mm/gg/aaaa*

Il valore dell'anno deve essere compreso tra 1980 e 2099. Se nell'indicare il giorno del mese o il mese stesso usate un numero di una cifra soltanto, verrà automaticamente posto uno zero davanti al numero stesso. Se per l'anno indicate solo una cifra, viene aggiunto uno zero per rendere il numero di due cifre; se le cifre date invece sono solo due, verrà assunto come anno il 19*aa* (*aa* sono ovviamente le uniche due cifre inserite).

## **DATE$** Variabile

Restituisce la data corrente.

*Sintassi:*
*varstr* = DATE$

*Esempi:*

```
PRINT DATE$
```

Viene restituita una stringa di 10 caratteri nella forma *mm-gg-aaaa*, ove *mm* rappresenta il mese scritto con un numero di due cifre, *gg* il giorno del mese, ancora in due cifre, infine *aaaa* l'anno. La data può anche essere inizializzata prima di entrare nel BASIC.

## **DEF FN** Istruzione

Definisce e dà un nome ad una funzione scritta dall'utente.

*Sintassi:*
DEF FN*nome*[(*argomento*[,*argomento*]...)] = *espressione*

*Esempio:*

```
DEF FNCHANGE(A,B)=A+RND*(B+1)
```

Il parametro *nome* deve essere un nome di variabile valido, e diventa, preceduto da FN, il nome della vostra funzione. *argomento*, è un nome di

variabile nella definizione della funzione che deve essere rimpiazzato con un valore ogni volta che la funzione viene richiamata. Gli argomenti della lista rappresentano, in base uno a uno, i valori che vengono assegnati quando è chiamata la funzione. Il parametro *espressione* definisce invece il valore che viene restituito dalla funzione, ed il suo tipo (numerico o stringa) deve essere uguale al tipo usato per *nome*.
La definizione della funzione è limitata ad una istruzione. Gli argomenti che appaiono nella definizione della funzione servono solo per definire la funzione stessa e non hanno alcun effetto sulle variabili del programma che portano lo stesso nome. Inoltre i nomi di variabili usati in *espressione* non devono necessariamente apparire nella lista degli argomenti: se ciò accade, il valore dell'argomento dev'essere fornito quando la funzione viene chiamata; in caso contrario, viene usato il valore corrente della variabile.
Il tipo della funzione determina se la funzione restituisce un valore numerico o una stringa, ed è dichiarato da *nome* allo stesso modo in cui le variabili vengono dichiarate. Se il tipo di *espressione* (numerico o stringa) non è uguale al tipo della funzione, si incorre nell'errore TYPE MISMATCH. Se la funzione è numerica, il valore dell'espressione viene convertito alla precisione specificata da *nome* prima di restituirlo all'istruzione che l'ha chiamata.
Un'istruzione DEF FN deve essere eseguita per definire una funzione prima che voi possiate chiamarla; se una funzione venisse chiamata prima di essere stata definita, si incorrerebbe in un errore UNDEFINED USER FUNCTION. D'altro canto, una funzione può essere definita più di una volta, ma viene utilizzata solo la più recente definizione eseguita.

NOTA: potreste trovarvi nella situazione di avere una funzione di tipo recursivo, cioè una funzione che chiama se stessa; se non provvederete a fermare in qualche modo la recursività, provocherete la visualizzazione di un messaggio d'errore OUT OF MEMORY.
DEF FN non è utilizzabile in modo diretto.

## **DEF SEG** Istruzione

Definisce il segmento corrente di memoria. Una istruzione BLOAD, BSAVE, CALL, PEEK, POKE o USR definirà l'indirizzo fisico corrente come offset all'interno del segmento.

*Sintassi:*
DEF SEG [= *indirizzo*]

*Esempio:*

```
DEF SEG = &HB800
```

Il parametro *indirizzo* è un numero appartenente all'intervallo da 0 a 65535.
L'inizializzazione per il segmento, quando il BASIC viene avviato, è il segmento dati (DS); esso rappresenta l'inizio della vostra area di lavoro all'interno della memoria del PC. Se eseguite un'istruzione DEF SEG che cambia il segmento, il valore non è reinizializzato al DS del BASIC quando inviate un comando RUN.
Se viene omesso *indirizzo* nell'istruzione DEF SEG, il segmento viene inizializzato al segmento dati BASIC; se invece viene passato un valore per *indirizzo*, dev'essere un multiplo di 16: il valore viene fatto slittare a destra di 4 byte (moltiplicato per 16) per costruire l'indirizzo del segmento della seguente operazione. Cioè, se l'indirizzo è in esadecimale, viene aggiunto uno zero per formare l'effettivo indirizzo del segmento.
Le parole DEF e SEG devono essere separate da uno spazio vuoto, altrimenti il BASIC interpreterebbe, ad esempio, l'istruzione DEFSEG=100 come un assegnamento del valore 100 alla variabile DEFSEG.
Ogni valore passato al di fuori dell'intervallo provocherà l'errore ILLEGAL FUNCTION CALL ed il mantenimento del precedente valore.

## **DEFtipo** Istruzione

Dichiara il tipo di una variabile come intera, a precisione semplice, a doppia precisione, stringa.

*Sintassi:*
DEF*tipo lettera*[ – *lettera*] [,*lettera*[ – *lettera*]]...

*Esempi:*

```
DEFINT A-C
DEFSNG M,Q
DEFDBL X,T,D-F
DEFSTR N-P
```

Il parametro *tipo* può essere INT, SNG, DBL, oppure STR; *lettera* invece è una lettera dell'alfabeto (dalla A alla Z). Un'istruzione DEF*tipo* dichiara che la variabile il cui nome comincia con la lettera o le lettere specificate sarà una variabile del tipo indicato. Comunque, un carattere di dichiarazione di tipo (%, !, #, $) avrà sempre la precedenza su un'istruzione DEF*tipo* nella tipizzazione della variabile.

Qualora non dovesse incontrare alcuna istruzione di dichiarazione di tipo, il BASIC considererà tutte le variabili prive del carattere di dichiarazione di tipo come variabili in precisione semplice. Se invece vengono usate le istruzioni di dichiarazione di tipo, queste dovranno essere poste all'inizio del programma. L'istruzione DEF*tipo* deve essere eseguita prima di usare una qualsivoglia variabile che essa stessa dichiara.

## **DEF USR** Istruzione

Specifica l'indirizzo di partenza delle subroutine in linguaggio macchina, che vengono successivamente chiamate dall'istruzione USR.

*Sintassi:*
DEF USR[*n*] = *offset*

*Esempio:*

```
DEF USR3=24000
```

Il parametro *n* identifica il numero della routine USR il cui indirizzo è stato specificato; quando viene omesso, è assunto per default DEF USR0.
Il parametro *offset* è un numero intero appartenente all'intervallo da 0 a 65535, il cui valore viene aggiunto al valore del segmento corrente per ottenere l'effettivo indirizzo di partenza della routine USR.
È possibile ridefinire l'indirizzo di una routine USR; in un programma può essere usato un numero illimitato di istruzioni DEF USR, permettendo così all'utente di accedere a tutte le routine che ritiene necessarie. Come *offset* viene usato il valore eseguito più di recente.

## **DELETE** (ALT D) Comando

Cancella linee di programma. Non valido in Compiled BASIC.

*Sintassi:*
DELETE [*linea1*] [–*linea2*]
DELETE [*linea1*–]

*Esempi:*

```
DELETE 40
DELETE 40-100
DELETE -40
```

Il parametro *linea1* rappresenta il numero della prima linea da cancellare; *linea2* invece è il numero di linea dell'ultima linea da cancellare; il comando DELETE cancella dal programma il gruppo specificato di linee. Si noti che il BASIC ritorna sempre al livello comandi dopo aver eseguito un comando DELETE. Il comando DELETE *linea1*– cancella tutte le linee a partire dalla linea specificata fino alla fine del programma.
Al posto del numero di linea può essere usato un punto per indicare che la linea che si intende cancellare è la linea corrente. Qualora venisse specificata una linea non esistente nel programma, si causerebbe un errore ILLEGAL FUNCTION CALL.

## **DIM** Istruzione

Specifica il valore massimo per gli indici di una variabile di tipo array e riserva di conseguenza lo spazio di memoria.

*Sintassi:*
DIM *variabile*(*indice*)[,*variabile*(*indice*)]...

*Esempi:*

```
DIM Q$(10,10),S(5),X(50)
```

Il parametro *variabile* è il nome da usare per l'array; *indice* è una lista di espressioni numeriche, separate da virgole, che definiscono le dimensioni dell'array.
Quando viene eseguita, l'istruzione DIM inizializza al valore 0 tutti gli elementi dello specificato array numerico, mentre gli elementi di un array-stringa sono tutti di lunghezza variabile, inizializzati alla lunghezza zero.
Se un nome di una variabile di tipo array viene usato senza l'istruzione DIM, il massimo valore del suo indice viene posto a 10. Qualora venisse usato un indice maggiore di quello massimo specificato, si provocherebbe la visualizzazione di un messaggio d'errore SUBSCRIPT OUT OF RANGE.
Il valore minimo per un indice è sempre 0, se non specificato altrimenti dall'istruzione OPTION BASE. Il numero massimo di dimensioni per un array è 255, ed il numero massimo di elementi per ogni dimensione è 32767. Entrambi questi numeri sono comunque limitati dalle dimensioni della memoria e dalla lunghezza delle istruzioni.
Se cercaste di dimensionare un array più di una volta, provochereste l'errore DUPLICATE DEFINITION; potete comunque usare l'istruzione ERASE per cancellare un array in modo da poterlo nuovamente dimensionare.

## DRAW Istruzione

Disegna un oggetto come specificato da *stringa*. È utilizzabile solo in Advanced BASIC ed in Compiled BASIC, in modo Grafico.

*Sintassi:*
DRAW "*stringa*"

*Esempi:*

```
DRAW "BM160,50 C2 G80 R160 H80"
DRAW "BM160,100 NU90 NE90 NR90 NF90 ND90 NL90 NH90"
DRAW "BM160,100 M-60,+30 M+20,-30 M-20,-30 M+60,+30"
DRAW DS$
```

Potete usare l'istruzione DRAW per disegnare usando un *linguaggio di definizione grafica*. I comandi di questo linguaggio sono contenuti in *stringa*. La stringa definisce un oggetto che viene disegnato quando il BASIC esegue l'istruzione DRAW. Durante l'esecuzione il BASIC esamina il valore di *stringa* ed interpreta i comandi a lettera singola dal contenuto di *stringa*. Questi comandi verranno spiegati in dettaglio in seguito.
I seguenti comandi di movimento fanno muovere il cursore a partire dall'ultima posizione occupata. Dopo ogni comando, l'ultima posizione occupata è l'ultima posizione del comando di disegno.

U *n* Muove il cursore verso l'alto
D *n* Muove il cursore verso il basso
L *n* Muove il cursore verso sinistra
R *n* Muove il cursore verso destra
E *n* Muove il cursore diagonalmente verso l'alto a destra
F *n* Muove il cursore diagonalmente verso il basso a destra
G *n* Muove il cursore verso il basso a sinistra
H *n* Muove il cursore verso l'alto a sinistra

In tutti i precedenti comandi *n* indica a quale distanza va portato il cursore. Il numero di punti di cui si muove è *n* volte il fattore di scala (evidenziabile tramite il comando S).

M *x,y* Movimento assoluto o relativo; se *x* ha un segno + o un segno − davanti, è relativo, altrimenti è assoluto.

Il rapporto tra le due dimensioni dello schermo determina la spaziatura dei punti orizzontali, verticali e diagonali. Per esempio, il rapporto standard di 4/3 indica che l'asse orizzontale dello schermo è 4/3 dell'asse ver-

ticale. Potete usare questa informazione per determinare il rapporto in lunghezza tra il numero di punti verticali ed il numero di punti orizzontali.
Per esempio, in media risoluzione vi sono 320 punti orizzontali e 200 punti verticali; questo significa che 8 punti orizzontali sono uguali in lunghezza a 5 punti verticali se il rapporto è 1/1. Se il rapporto è differente, moltiplicate il numero di punti verticali per il rapporto stesso. Usando il rapporto standard, che come abbiamo appena visto è 4/3, in media risoluzione 8 punti orizzontali sono in lunghezza uguali a 20/3 punti verticali, oppure 24 punti orizzontali sono uguali a 20 punti verticali. Questo significa che:

```
DRAW "U80 R96 D80 L96"
```

disegna un quadrato in media risoluzione. Seguendo un ragionamento del tutto analogo, sempre con uno schermo il cui rapporto sia 4/3, in alta risoluzione 48 punti orizzontali sono uguali a 20 punti verticali.
I seguenti due prefissi possono precedere ognuno dei precedenti comandi:

B Muove il cursore ma non disegna nessun punto

N Muove il cursore ma al termine ritorna alla posizione di partenza.

Sono inoltre disponibili i seguenti comandi:

A *n* Inizializza l'angolo *n*, che può assumere valori da 0 a 3, dove 0 equivale a 0 gradi, 1 equivale a 90 gradi, 2 equivale a 180 gradi e 3 equivale a 270 gradi. Le figure ruotate di 90 o 270 gradi sono dimensionate in modo da apparire della stessa misura di quando sono orientate a 0 o 180 gradi su di uno schermo con rapporto 4/3.

TA *n* Ruota di un angolo *n*, il cui intervallo di valori è da −360 a +360 gradi. Se *n* è positivo ruota in senso antiorario: se *n* è negativo ruota in senso orario. Introducendo un valore al di fuori dell'intervallo consentito si causerà l'errore ILLEGAL FUNCTION CALL.

C *n* Inizializza al colore *n*, con valori da 0 a 3 in media risoluzione e da 0 a 1 in alta risoluzione. In media risoluzione *n* seleziona il colore dalla tavolozza corrente come definita dall'istruzione COLOR; 0 è il colore di fondo; il valore per il colore di testo è 3. In alta risoluzione $n=0$ indica i colore nero ed il valore di default $n=1$ indica il bianco.

S *n* Inizializza il fattore di scala; *n*, il cui intervallo di variabilità è da 0 a 255, rappresenta il fattore di scala. Per esempio se

$n$=1 il fattore di scala è 1/4. Il fattore di scala moltiplicato per la distanza data con i comandi U, D, L, R, E, F, G, H ed il comando M relativo (movimento in coordinate relative) dà l'effettivo spostamento. Il valore di default è 4, per il quale il fattore di scala è 1.

X *stringa*;

Esegue sottostringhe, permette cioè di eseguire stringhe dall'interno di altre.

P *colore,contorno*

Inizializza il colore della figura a *colore* ed il colore del bordo a *contorno*. Il parametro *colore*, che può variare tra 0 e 3, in media risoluzione è il colore della tavolozza corrente come definita dall'istruzione COLOR; in alta risoluzione 0 indica nero ed 1 indica bianco. Il parametro *contorno* è il colore del bordo della figura da riempire: l'intervallo di colori è, come per *colore*, da 0 a 3. Entrambi questi parametri sono necessari ed omettendone anche uno solo si incorrerebbe in un errore. Non è possibile la pittura a motivi (vedi PAINT).

In tutti questi comandi gli argomenti *n*, *x* e *y* possono essere delle costanti oppure variabili numeriche, purché seguite da un punto e virgola, nella forma =*variabile*;. Il punto e virgola è richiesto quando si usa una variabile in questo modo, o nel comando X. Inoltre è possibile porre un punto e virgola tra due comandi. Gli spazi all'interno delle stringhe vengono ignorati. Per esempio potreste usare le variabili in un comando di movimento in questo modo:

```
M+=X1;,-=X2;
```

Potete anche specificare le variabili nella forma VARPTR$(*variabile*), invece di =*variabile*;. Questo è utile in programmi che saranno compilati più avanti. Per esempio:

| **Primo metodo** | **Metodo alternativo** |
|---|---|
| DRAW"XA$;" | DRAW"X"+VARPTR$(A$) |
| DRAW"S=SCALE;" | DRAW"S="+VARPTR$(SCALE) |

Il comando X può essere una parte molto utile in un'istruzione DRAW poiché potete definire una parte di un oggetto separatamente dal resto: ad esempio un sorriso su un volto. Potete inoltre usare X per disegnare una stringa di comandi lunga più di 255 caratteri.

## **EDIT** Comando

Visualizza una linea per l'inserimento di caratteri. Non è utilizzabile in Compiled BASIC.

*Sintassi:*
EDIT *linea*

*Esempio:*

```
EDIT 110
```

Il parametro *linea* deve essere il numero di linea di una linea esistente nel programma, in caso contrario verrà visualizzato un messaggio d'errore UNDEFINED LINE NUMBER. Al posto di *linea* può essere usato un punto quando si intende fare riferimento alla linea corrente. Per esempio, se voi avete appena inserito in memoria una linea di programma ma la volete riportare sullo schermo per correggerla, vi basterà dare il comando EDIT per vederla ricomparire. Il comando EDIT può quindi essere utilizzato per visualizzare linee di programma cui apportare modifiche. L'istruzione EDIT visualizza semplicemente la linea specificata e posiziona il cursore sotto il primo carattere del numero di linea; a questo punto la linea può essere modificata utilizzando i tasti INS, DEL e di controllo cursore.

## **END** Istruzione

Termina l'esecuzione del programma, chiude tutti i file, e ritorna al livello comandi.

*Sintassi:*
END

*Esempio:*

```
END
```

L'istruzione END può essere inserita in un qualsiasi punto del programma per terminare l'esecuzione; essa ha due differenze sostanziali rispetto all'istruzione STOP:

— non provoca la visualizzazione del messaggio di interruzione
— chiude tutti i file

L'uso dell'istruzione END al termine del programma è opzionale. Dopo l'esecuzione di END il BASIC torna subito al livello comandi.

## **EOF** Funzione

Indica la condizione di end-of-file (EOF).

*Sintassi:*
*varnum* = EOF(*numfile*)

*Esempio:*

```
IF EOF(1) THEN 1000
```

Il parametro *numfile* è il numero che è stato usato nell'istruzione OPEN. La funzione EOF è utile per evitare di incorrere nell'errore INPUT PAST END; essa restituisce il valore −1 (vero) se nel file specificato è stato incontrato un EOF, altrimenti il valore restituito è uno 0.
EOF è una funzione significativa solo per un file che sia stato aperto per input di tipo sequenziale da disco, oppure per file rivolti alla comunicazione. Ad esempio, per un file rivolto alla comunicazione una risposta uguale a −1 significa che il buffer di comunicazione è vuoto.
In particolare EOF(0) restituisce la condizione di EOF su dispositivi di input standard usati con redirezione di I/O.

## **ERASE** Istruzione

Cancella gli array da un programma. Non è utilizzabile in Compiled BASIC.

*Sintassi:*
ERASE *nomearray*[,*nomearray*]...

*Esempio:*

```
ERASE A,B
```

Il parametro *nomearray* è il nome dell'array che si desidera cancellare. Potrebbe esservi utile usare l'istruzione ERASE nel caso in cui vi accorgeste di avere poca memoria a disposizione durante l'esecuzione di un programma. Dopo che l'array è stato cancellato, lo spazio di memoria che gli era stato riservato può essere usato per altri scopi.

L'istruzione ERASE può anche essere usata per ridimensionare gli array nei programmi; infatti se cercaste di ridimensionare un array senza averlo prima cancellato, incorrereste nell'errore DUPLICATE DEFINITION. Simile è il comando CLEAR, che viene usato per cancellare tutte le variabili dall'area di lavoro.

## **ERR e ERL** Variabili

Restituiscono il codice d'errore ed il numero di linea associato all'errore.

*Sintassi:*
*varnum* = ERR
*varnum* = ERL

*Esempi:*

```
IF ERR=24 THEN RESUME
M=ERL
```

La variabile ERR contiene il codice dell'ultimo errore, e la variabile ERL contiene il numero della linea in cui l'errore è stato trovato; sono entrambe solitamente usate all'interno delle istruzioni IF-THEN per dirigere l'andamento del programma nella routine di gestione dell'errore (vedi ON ERROR).
Se introducete ERL in un'istruzione IF-THEN, fate molta attenzione a porre il numero di linea a destra dell'operatore relazionale, come nell'esempio seguente:

IF ERL = *linea* THEN...

infatti, così facendo, il numero può essere cambiato da RENUM.
Se l'istruzione che ha causato l'errore era in modo diretto, la variabile ERL conterrà 65535. Poiché non si vuole che questo numero sia cambiato dall'istruzione RENUM, se si desidera controllare in quale errore sia incorsa l'istruzione in modo diretto basterà usare la forma:

IF 65535 = ERL THEN...

Infine le variabili ERR e ERL possono essere inizializzate usando l'istruzione ERROR.

## ERROR Istruzione

Simula un errore BASIC o vi permette di definire un vostro proprio codice d'errore.

*Sintassi:*
ERROR *n*

*Esempi:*

```
ERROR T
IF B>5000 THEN ERROR 210
```

Il parametro *n* deve essere un'espressione intera compresa tra 0 e 255. Se il valore di *n* è lo stesso del codice di un errore usato dal BASIC, l'istruzione ERROR simula quell'errore. La routine di errore verrà eseguita se è stata definita una routine di intercettamento degli errori per mezzo dell'istruzione ON ERROR. In caso contrario verrà visualizzato il messaggio d'errore corrispondente al codice e verrà interrotta l'esecuzione del programma in corso.
Per definire un vostro proprio codice d'errore, dovete usare un valore che sia differente da ogni altro usato dal BASIC (vi consigliamo di usare i più alti valori disponibili, ad esempio quelli maggiori di 200). Questo nuovo codice d'errore può quindi essere controllato da una routine di gestione degli errori, proprio come avviene per gli altri errori. Nel caso in cui definiste i vostri propri codici d'errore nel modo appena descritto ma senza farli gestire da una apposita routine, il BASIC visualizzerà il messaggio UNPRINTABLE ERROR, interrompendo l'esecuzione in corso.

## EXP Funzione

Calcola la funzione esponenziale.

*Sintassi:*
*varnum* = EXP(*esprnum*)

*Esempio:*

```
Q=EXP(X)
```

## **FIELD** Istruzione

Riserva spazio per le variabili in un buffer per file ad accesso diretto.

*Sintassi:*
FIELD [#] *numfile, dim* AS *varstr* [, *dim* AS *varstr*]...

*Esempio:*

```
FIELD 1,20 AS N$,10 AS ID$
```

Il parametro *numfile* è il numero con cui il file era stato aperto; *dim* è un'espressione numerica al massimo uguale a 255 che specifica il numero di posizioni per i caratteri da riservare per *varstr*; quest'ultima è una variabile stringa che verrà usata per accedere al file ad accesso diretto.
Un'istruzione FIELD definisce le variabili che vengono usate per estrarre dati da un buffer ad accesso diretto dopo un'istruzione GET o per inserire dati nel buffer per un'istruzione PUT.
L'istruzione:

```
FIELD 1,20 AS N$,10 AS ID$,40 AS ADD$
```

riserva le prime 20 posizioni (byte) nel file ad accesso diretto del buffer per la variabile stringa N$, le seguenti 10 posizioni per ID$, e le ultime 40 posizioni per ADD$. L'istruzione non immette i dati nel file ad accesso diretto del buffer; questo compito viene svolto dalle istruzioni LSET e RSET.
L'istruzione FIELD non è neppure in grado di eliminare i dati dal file: le informazioni vengono trasferite dal file al buffer con l'istruzione GET e vengono poi lette dal buffer semplicemente facendo riferimento alle variabili definite nell'istruzione FIELD.
Il numero totale di byte riservati con un'istruzione FIELD non deve superare la lunghezza del record definita quando il file è stato aperto, altrimenti si verifica un errore FIELD OVERFLOW.
Un qualsiasi numero di istruzioni FIELD può essere eseguito per lo stesso file, e tutte le istruzioni FIELD eseguite sono valide contemporaneamente; ogni nuova istruzione FIELD ridefinisce il buffer a partire dalla prima posizione e ciò ha in pratica l'effetto di assegnare diverse definizioni di campo per gli stessi dati.

NOTA: fate attenzione a non usare in un'istruzione di input o di assegnamento il nome di una variabile già inserita in un'istruzione

FIELD. Infatti dopo la definizione in FIELD la variabile punta alla posizione corretta all'interno del buffer; se viene eseguita successivamente un'istruzione di input o un'istruzione LET in cui quel nome di variabile compaia a sinistra del segno di uguale, la variabile viene portata nello spazio delle stringhe e non si trova più nel buffer.

## **FILES** Comando

Visualizza i nomi dei file residenti su un disco. Il comando FILES in BASIC è simile al comando DIR in DOS.

*Sintassi:*
FILES ["*specfile*"]

*Esempi:*

```
FILES
FILES "*.BAS"
FILES "TEST??.BAS"
FILES "A:"
FILES "\PRIMO\.*.BAS"
```

Quando il parametro *specfile* viene omesso, si ottiene l'elenco di tutti i file presenti sul disco di default del DOS e la visualizzazione del catalogo corrente.
I file visualizzati sono tutti quelli con il nome corrispondente a quello indicato; al riguardo si osservi che il nome o l'estensione possono contenere un punto interrogativo, che viene considerato dal BASIC uguale a qualsiasi carattere; mentre un asterisco come primo carattere del nome o dell'estensione vale per un qualsiasi nome o estensione.
Se all'interno di *specfile* viene indicato il drive cui riferire il comando, viene visualizzata la lista dei file il cui nome è uguale a quello contenuto in *specfile* e che si trovano nel disco contenuto nel drive indicato; in caso contrario viene usato il drive di default.

## **FIX** Funzione

Tronca *esprnum* ad un numero intero.

*Sintassi:*
*varnum* = FIX(*esprnum*)

*Esempio:*

```
PRINT FIX (45.67)
 45
```

Questa funzione elimina tutte le cifre alla destra della virgola decimale e restituisce il valore delle cifre alla sinistra della virgola. La differenza tra FIX e INT è che FIX non restituisce il successivo valore più basso se *epr-num* è negativa. Vedere anche INT e CINT per la riduzione ad interi.

## **FOR** (ALT A) - **NEXT** (ALT N) Istruzione

Esegue ciclicamente una serie di istruzioni per un dato numero di volte.

*Sintassi:*

FOR *variabile* = *x* TO *y* [STEP *z*]
:
NEXT [*variabile*] [,*variabile*]...

*Esempi:*

```
FOR K=3 TO T
.
.
.
NEXT K
FOR M=100 TO 1 STEP -1
.
.
.
NEXT
```

Il parametro *variabile* è un intero in precisione semplice o una variabile in doppia precisione da usare come contatore; *x* è un'espressione numerica che rappresenta il valore iniziale per il contatore; *y* è un'espressione numerica che rappresenta il valore finale per il contatore; *z* è un'espressione numerica che rappresenta il valore dell'incremento per il contatore. Le linee di programma che seguono l'istruzione FOR vengono eseguite fino a che non si incontra l'istruzione NEXT; a questo punto il contatore viene incrementato della quantità specificata dal valore di STEP, che viene posto per default a 1 quando non è specificato altrimenti. Viene ora eseguito un test per controllare se il valore del contatore ha superato il valore finale *y*; se questo non si verifica il BASIC salta indietro all'istruzione che segue l'istruzione FOR ed il processo viene ripetuto. Quando

poi il contatore sarà maggiore di *y*, l'esecuzione continuerà con l'istruzione che segue NEXT. Questo processo va sotto il nome loop FOR-NEXT.
Se il valore di *z* è negativo, il test avviene al rovescio: il contatore viene decrementato ad ogni loop, ed il loop viene eseguito fino a che il contatore non è minore del valore finale.
Il contenuto del loop viene completamente saltato se *x* è già maggiore di *y* quando STEP è positivo, oppure se *x* è minore di *y* quando STEP è negativo. Se poi *z* è uguale a zero, si incorre nel caso di loop infinito, a meno che l'utente non provveda in qualche modo a porre il contatore maggiore del valore finale.
L'esecuzione di programmi sarà senz'altro più precisa e veloce se si useranno, ove possibile, dei numeri interi come contatori.

**Loop annidati**

I loop FOR-NEXT possono essere annidati: ciò significa che un loop FOR-NEXT può essere posto all'interno di un altro loop FOR-NEXT. Quando dei loop sono annidati, ogni loop deve avere un proprio nome di variabile come contatore. Inoltre l'istruzione NEXT del loop più interno deve essere posta prima di quella del loop più esterno; se però dei loop annidati hanno lo stesso punto di fine, può essere usata per tutti la stessa istruzione NEXT.
Un'istruzione NEXT di forma:

NEXT *var1, var2, var3 ...*

è equivalente alla seguente sequenza di istruzioni:

NEXT *var1*
NEXT *var2*
NEXT *var3*

La variabile o le variabili nell'istruzione NEXT possono essere omesse, nel qual caso l'istruzione NEXT serve per l'istruzione FOR più recente. Se avete usato dei loop FOR-NEXT annidati, dovrete includere le variabili in tutte le istruzioni NEXT. Ciò permette di evitare confusioni, ma diventa necessario se fate dei salti al di fuori dei loop annidati. Comunque, l'utilizzo dei nomi delle variabili nelle istruzioni NEXT causerà un certo rallentamento nell'esecuzione del programma.
Nel caso in cui vi fosse un'istruzione NEXT prima della corrispondente istruzione FOR, si incorrerebbe nell'errore NEXT WITHOUT FOR.

## FRE Funzione

Restituisce il numero di byte in memoria che non sono stati utilizzati dal BASIC. Questo numero non comprende la dimensione dello spazio riservato come area di lavoro dell'interprete (solitamente da 2.5KB a 4KB).

*Sintassi:*
*varnum* = FRE(*esprnum*)
*varnum* = FRE(*stringa*)

*Esempio:*

```
PRINT FRE(X)
```

I parametri *esprnum* e *stringa* sono argomenti fittizi. Poiché in BASIC le stringhe possono avere lunghezze variabili (ogni volta che voi fate un assegnamento ad una stringa la sua lunghezza può cambiare), vengono manipolate dinamicamente. Per questa ragione lo spazio per le stringhe può diventare frammentato.
La funzione FRE con ogni valore *stringa* provoca un riordino prima di restituire il numero di byte; il riordino consiste nella cancellazione di tutti gli spazi non utilizzati, nella ricucitura di tutti i blocchi frammentati e nel ricupero della memoria utilizzata da variabili e stringhe non più necessarie. I dati vengono compressi cosicché potete continuare finché non andate oltre lo spazio di lavoro.
Il BASIC compie automaticamente il riordino ogni volta che arriva al limite dell'area di lavoro. Voi potreste usare FRE (" ") periodicamente per rendere più brevi i tempi di ogni operazione di riordino.
Il comando CLEAR inizializza il massimo numero di byte come area di lavoro del BASIC. FRE restituisce la quantità di memoria libera nell'area di lavoro del BASIC. Se nell'area di lavoro non vi è nulla, allora il valore restituito da FRE sarà di 2.5KB o 4KB (le dimensioni dell'area di lavoro riservata per l'interprete) inferiore al totale della memoria disponibile.

## GET (File) Istruzione

Legge un record da un file ad accesso diretto o dal buffer di comunicazione.

*Sintassi:*
GET[#]*numfile*[,*numero record*]

*Esempi:*

```
GET 1
GET #2,5
```

Il parametro *numfile* è il numero con cui il file era stato aperto; *numero record* è il numero di record da leggere, ed è compreso nell'intervallo da 0 a 16 777 215; se *numero record* viene omesso, viene copiato nel buffer il record successivo, dopo l'ultimo GET.
Dopo un'istruzione GET, possono essere usati INPUT#, LINE INPUT#, o un riferimento ad una variabile definita in un'istruzione FIELD, per leggere caratteri dal buffer del file ad accesso casuale. Poiché il BASIC ed il DOS racchiudono il maggior numero possibile di record in settori da 512 byte, l'istruzione GET non necessariamente compie una lettura fisica dal disco.
GET può inoltre essere usata per file di comunicazioni. In questo caso *numero record* è il numero di byte da leggere dal buffer di comunicazione. Questo numero non può superare il valore inizializzato dall'opzione LEN nell'istruzione OPEN"COM...

## **GET** (Grafica) Istruzione

Legge punti da un'area dello schermo. Utilizzabile in Advanced BASIC ed in Compiled BASIC, solo in modo Grafico.

*Sintassi:*
GET(*x1,y1*)−(*x2,y2*),*vettoreimmagine*

*Esempio:*

```
GET (50,50)-(59,62), D$
```

(*x1,y1*) e (*x2,y2*) sono le coordinate sia in forma relativa che in forma assoluta; *vettoreimmagine* è il nome dell'array in cui volete mettere le informazioni.
L'istruzione GET legge il colore dei punti all'interno del rettangolo specificato nell'array; questo rettangolo ha i punti (*x1,y1*) e (*x2,y2*) ad angoli opposti (è l'equivalente di disegnare il rettangolo con l'istruzione LINE e l'opzione B).
Le istruzioni GET e PUT possono essere usate per il movimento di oggetti ad alta velocità in alta risoluzione. Potreste ad esempio usare GET e PUT alternativamente per simulare il movimento di un bit sullo schermo.
Si ricordi che GET e PUT vengono anche usate per file ad accesso diretto, ma la sintassi di queste istruzioni è differente.

L'array è usato semplicemente come contenitore della figura, e deve essere numerico, non importa se in doppia o semplice precisione. Le dimensioni richieste per l'array sono, in byte:

4+INT ((*colonne* * (*2/modo*)+7)/8) * *righe*

dove:
— *colonne* è il numero di colonne (larghezza dell'area)
— *modo* è il modo Grafico, 1 o 2
— *righe* è il numero di righe di schermo (altezza dell'area)

Per esempio, supponiamo di voler usare l'istruzione GET per voler ottenere una figura 10×12 in media risoluzione. Il numero di byte richiesti è:

4+INT ((10*2+7)/8)*12 = 40.

I byte per elemento di un array sono:

— Due per gli interi
— Quattro per la precisione semplice
— Otto per la doppia precisione

Dovremo usare, perciò, un array di interi con almeno 20 elementi.
Le informazioni dello schermo vengono immagazzinate nell'array nel modo seguente:

— Due byte che danno la dimensione $x$ in bit
— Due byte che danno la dimensione $y$ in bit
— I dati

È possibile esaminare le dimensioni di $x$ e $y$ e perfino i dati stessi se viene usato un array di interi. La dimensione di $x$ è nell'elemento 0 dell'array e la dimensione di $y$ nell'elemento 1. Ricordate, comunque, che gli interi sono immagazzinati memorizzando prima il byte meno significativo, poi il byte più significativo, mentre questi dati vengono in realtà trasferiti con prima il byte più significativo e poi quello meno significativo.
Il dato per ogni insieme di punti nel rettangolo è giustificato a sinistra sul byte di contorno, cosicché se questi sono minori di un multiplo degli otto bit immagazzinati, il resto del byte sarà riempito di zeri.
Le istruzioni GET e PUT lavorano in maniera significativamente più veloce in media risoluzione quando *x1* MOD 4 è uguale a zero ed in alta risoluzione quando *x1* MOD 8 è uguale a zero; questo è un caso particolare in cui i contorni del rettangolo coincidono con i byte di contorno.

## **GOSUB** Istruzione

Trasferisce il controllo a una subroutine.

*Sintassi:*
GOSUB *linea*
:
RETURN

*Esempio:*

```
GOSUB 6000
.
.
.
6000 REM SUBROUTINE DEL JOYSTICK
.
.
.
RETURN
```

Il parametro *linea* è il numero della prima linea della subroutine. Una subroutine può essere chiamata all'interno di un programma tante volte quante si vuole, e si possono fare chiamate anche dall'interno di altre subroutine; la possibilità di usare subroutine annidate è limitata solo dalla quantità di memoria disponibile.
L'istruzione RETURN indica al BASIC di ritornare all'istruzione che segue immediatamente quella che contiene la più recente istruzione GOSUB. Una subroutine può contenere più di una istruzione RETURN se si desidera ritornare al punto di chiamata da differenti punti. Le subroutine possono essere messe in un qualunque punto del programma.
Per evitare delle chiamate accidentali a subroutine, potete mettere un'istruzione STOP, END, o GOTO davanti alla subroutine per permettere al programma di fare dei controlli.
Infine si può usare l'istruzione ON-GOSUB per saltare a differenti subroutine in base al risultato di una espressione.

## **GOTO** (ALT G) Istruzione

Compie un salto incondizionato ad uno specificato numero di linea al di fuori della sequenza normale del programma.

*Sintassi:*
GOTO *linea*

*Esempio:*

```
GOTO 100
```

Il parametro *linea* è il numero di una linea all'interno del programma, e se corrisponde ad un numero di linea contenente un'istruzione eseguibile, quell'istruzione e le seguenti vengono normalmente eseguite; se invece corrisponde ad un numero di linea contenente un'istruzione non eseguibile (come ad esempio REM o DATA), il programma continua dalla successiva istruzione eseguibile incontrata.
L'istruzione GOTO può essere usata in modo diretto per far ripartire un programma da un punto specificato, cosa che può essere utile nella fase di debugging. Infine si può usare l'istruzione ON-GOTO per saltare a differenti subroutine basate sul risultato di una espressione.

## **HEX$** (ALT H) Funzione

Restituisce una stringa che rappresenta il valore in esadecimale di un argomento decimale.

*Sintassi:*
*vastr* = HEX$(*esprnum*)

*Esempio:*

```
PRINT HEX$(100)
```

Il parametro *esprnum* è un numero compreso nell'intervallo da −32768 a +65535; quando è negativo, viene eseguito il suo complemento a due, cioè HEX$(*esprnum*) è equivalente a HEX$(*esprnum*−65536). Vedere la funzione OCT$ per la conversione in ottale.

## **IF** Istruzione

Prende una decisione riguardante l'andamento del programma, basandosi sul risultato di un'espressione.

*Sintassi:*
IF *espressione* [,] THEN *proposizione* [ELSE *proposizione*]
IF *espressione* [,] GOTO *linea* [[,]ELSE *proposizione*]

*Esempi:*

```
IF A<5.5 THEN ERROR 200
IF M<>N GOTO 20, ELSE STOP
IF X THEN PRINT "Termine diverso da zero"
THEN PRINT "X=";:BEEP:ELSE PRINT "X<>0"
```

Il parametro *proposizione* può essere un'istruzione BASIC o una sequenza di istruzioni separate dai due punti, oppure semplicemente il numero di una linea cui saltare; *linea* è il numero di una linea esistente nel programma.
Se *espressione* è vera (cioè non zero), allora viene eseguita la proposizione THEN o GOTO. THEN può essere seguita sia da una linea cui saltare sia da una o più istruzioni.
Se invece *espressione* è falsa (cioè uguale a zero), allora le proposizioni di THEN o di GOTO vengono ignorate, e viene eseguita, se presente, la proposizione di ELSE. L'esecuzione continua con la successiva istruzione eseguibile.

NOTA: quando si usa l'istruzione IF per verificare l'uguaglianza di un valore che è il risultato di un calcolo eseguito in semplice o in doppia precisione, ricordate che la rappresentazione interna del valore può non essere esatta (questo accade perché i valori in semplice e doppia precisione vengono memorizzati internamente in formato a virgola mobile). Perciò, il test deve controllare l'intero intervallo in cui può variare l'accuratezza del numero. Per esempio, per controllare il valore calcolato per la variabile A rispetto al valore 1.0, si usa:

```
IF ABS(A-1.0)<1.0E-6 THEN...
```

Questo test restituisce "vero" se il valore di A è 1.0 con un errore relativo minore di 1.0E−6.
Inoltre si noti che IF-THEN-ELSE sono un sola istruzione, cioè la clausola ELSE non può essere una linea a sé stante. Per esempio, in

```
10 IF A=B THEN X=4
20 ELSE P=Q
```

la linea 20 non sarà mai eseguita; infatti la forma corretta sarebbe:

```
10 IF A=B THEN X=4 ELSE P=Q
```

**Annidamento di istruzioni IF-THEN-ELSE**

Le istruzioni IF-THEN-ELSE possono essere annidate, ma con la limitazione che l'annidamento può essere fatto solo per la lunghezza di una linea. Per esempio:

```
IF X>Y THEN PRINT "MAGGIORE DI" ELSE IF Y>X THEN
 PRINT "MINORE DI" ELSE PRINT "UGUALE A"
```

è un'istruzione valida. Se l'istruzione non contiene lo stesso numero di THEN e ELSE, ogni ELSE corrisponde al THEN più vicino. Per esempio:

```
IF A=B THEN IF B=C THEN PRINT "A=C"
  ELSE PRINT "A<>C"
```

non stamperà "A< >C" quando è A< >B.
Il significato degli IF annidati può essere chiarito con la formattazione. Per esempio:

```
100 IF ID$<"M" THEN IF D=1 GOTO 1000 ELSE GOSUB 6000
       ELSE D=1
```

può essere riscritta:

```
100 IF ID$<"M" THEN IF D=1 GOTO 1000
                        ELSE GOSUB 6000
       ELSE D=1
```

Per ottenere un ritorno a capo del cursore senza memorizzare la linea, usate CTRL ENTER.

## INKEY$ Variabile

Legge un carattere dalla tastiera.

*Sintassi:*
*varstr* = INKEY$

*Esempio:*

```
100 A$=INKEY$:IF A$="" THEN 100
```

INKEY$ legge solamente un singolo carattere, anche se nel buffer della

tastiera vi sono più caratteri in attesa. Il valore restituito può essere una stringa di zero, uno, o due caratteri.

— Una stringa nulla indica che non vi sono caratteri in attesa dalla tastiera.
— Una stringa di un carattere contiene il carattere appena letto dalla tastiera.
— Una stringa di due caratteri indica uno speciale codice esteso. Il primo carattere sarà l'esadecimale 00 (vedi Appendice B per la lista completa dei codici).

Poiché usando la variabile INKEY$ il suo valore viene portato al valore "stringa nulla", dovete assegnare il risultato di INKEY$ ad una variabile stringa prima di usare il carattere con una qualsiasi istruzione o funzione BASIC.
Mentre viene usata la variabile INKEY$, sullo schermo non è visualizzato nessun carattere e tutti i caratteri vengono passati direttamente al programma, eccetto:

— CTRL BREAK, che arresta il programma
— CTRL NUM LOCK, che manda il programma in uno stato di pausa
— ALT CTRL DEL, che reinizializza il sistema
— PRTSC, che stampa lo schermo.

Se in risposta alla variabile INKEY$ premete il tasto ENTER, al programma viene passato direttamente il carattere di ritorno carrello.

## **INP** Funzione

Restituisce il byte letto dalla porta *esprnum.*

*Sintassi:*
*varnum* = INP(*esprnum*)

Il parametro *esprnum* deve essere compreso nell'intervallo da 0 a 65535. La funzione INP è complementare all'istruzione OUT ed ha lo stesso compito dell'istruzione IN del linguaggio Assembler. Vedere il manuale IBM *Personal Computer Technical Reference* per l'elencazione dei numeri validi di porte di input (indirizzi di I/O).

## **INPUT** (ALT I) Istruzione

Riceve input da tastiera durante l'esecuzione di un programma.

*Sintassi:*

INPUT[;] ["*stringa prompt*";]*variabile*[,*variabile*]...

*Esempi:*

```
INPUT N
INPUT "Primo nome: ", NA$
INPUT; "Coordinate: X: ", X: INPUT "Y: .",Y
INPUT "Ne vuoi un altro "; Y$
```

Il parametro *stringa prompt* è una stringa che sarà usata come prompt per gli input desiderati; *variabile* è il nome di una variabile numerica o a stringa, oppure di un elemento di array che ha la funzione di ricevere l'input.
Quando il programma incontra un'istruzione INPUT, si arresta e visualizza sullo schermo un punto interrogativo per indicare che sta aspettando dei dati; se è stato inserito il parametro *stringa prompt*, la stringa sarà visualizzata prima del punto interrogativo. A questo punto potete inviare da tastiera i dati richiesti.
Se poi desiderate che il punto interrogativo non venga visualizzato, vi basta inserire una virgola invece di un punto e virgola dopo la *stringa prompt* per sopprimerlo; per esempio l'istruzione

```
INPUT "INSERIRE DATA DI NASCITA ",B$
```

scrive il prompt senza punto interrogativo.
Il dato che voi inserite viene assegnato alla variabile (o alle variabili se i dati sono più d'uno) inserita nella lista delle variabili; questi dati devono essere separati da virgole ed il loro numero deve coincidere col numero di variabili presenti nella lista.
Il tipo di ogni dato deve essere lo stesso della corrispondente variabile della lista. (Le stringhe inserite in risposta ad ogni istruzione INPUT non devono necessariamente essere separate da virgolette, a meno che non contengano delle virgole o degli spazi bianchi significativi).
Se ad un'istruzione INPUT rispondete con troppi o troppo pochi termini o con un tipo sbagliato di valori (lettere al posto di numeri, per esempio), il BASIC visualizzerà il messaggio ?REDO FROM START. Se viene richiesta una singola variabile, potete semplicemente premere il tasto ENTER per indicare che il vostro input è il valore di default, 0, per l'input numerico e la stringa nulla per l'input di tipo stringa. Per assegnare da tastiera il valore di default ad una lista di variabili, è sufficiente non introdurre nulla tra le virgole. Per esempio,

```
1,,3,
```

soddisfa un'istruzione INPUT per 4 variabili; la seconda e la quarta va-

riabile avranno il valore di default. Il BASIC non assegnerà nessuno dei valori di input finché non darete una risposta accettabile.
In Disk ed in Advanced BASIC, se l'istruzione INPUT è immediatamente seguita da un punto e virgola, premendo il tasto ENTER per introdurre dati, sullo schermo non si verificherà il ritorno a capo: ciò significa che il cursore rimane sulla stessa linea della vostra risposta.

## **INPUT#** Istruzione

Legge dati da un dispositivo o file sequenziale e li assegna alle variabili del programma.

*Sintassi:*

INPUT# *numfile, variabile*[,*variabile*]...

*Esempio:*

```
INPUT #1,A$,B
```

Il parametro *numfile* è il numero usato al momento dell'apertura del file; *variabile* è il nome della variabile cui verranno assegnati i termini del file, e potrà essere una variabile stringa o numerica o anche un array di variabili.
Il file sequenziale può risiedere su disco, o essere una lista di dati sequenziali provenienti da un'interfaccia di comunicazione, o essere la tastiera stessa (KYBD:).
Il tipo di dati del file deve essere uguale a quello specificato nel nome della variabile. A differenza dell'istruzione INPUT, nel caso di INPUT# non viene visualizzato il punto interrogativo.
I dati del file sequenziale devono essere disposti nello stesso modo in cui apparirebbero se i dati stessi fossero stati inviati in risposta ad un'istruzione INPUT. Con i valori numerici vengono ignorati gli spazi iniziali, i ritorni carrello e i ritorni a capo. Il primo carattere incontrato viene assunto come carattere di inizio dei numeri, a meno che non sia uno spazio iniziale significativo, un ritorno carrello, o un a capo. Il numero termina con uno spazio, un ritorno carrello, un a capo o una virgola.
Se il BASIC sta esplorando i dati alla ricerca di un termine di tipo stringa, gli spazi iniziali, i ritorni carrello e gli a capo vengono ancora ignorati; il primo carattere incontrato che non sia uno dei tre precedenti sia assunto come inizio per la stringa. Se il primo carattere sono le virgolette, la stringa sarà composta da tutti i caratteri letti tra il primo ed il secondo simbolo delle virgolette; così una stringa racchiusa tra virgolette non può contenere altre virgolette. Se invece il primo carattere della stringa

non sono le virgolette, la stringa viene considerata una stringa non racchiusa tra virgolette ed avrà termine quando si leggerà una virgola, un ritorno carrello, un a capo, oppure quando saranno stati letti 255 caratteri. Se infine viene letto un carattere EOF quando è stato ricevuto in input un termine numerico o un termine di tipo stringa, il termine stesso verrà cancellato.
L'istruzione INPUT# può anche essere usata con i file ad accesso diretto.

## **INPUT$** Funzione

Restituisce una stringa di caratteri letta da tastiera o da un file di numero *numfile*.

*Sintassi:*
*vastr* = INPUT$(*n*[,[#]*numfile*])

*Esempi:*

```
V$=INPUT$ (10)
N$=INPUT$ (80,#2)
IF INPUT$ (1)="N" THEN END
```

Il parametro *n* è il numero di caratteri da leggere dalla tastiera o dal file; *numfile* è il numero usato nell'istruzione OPEN; quando viene omesso, i dati sono letti dalla tastiera.
Se per l'input viene usata la tastiera, sullo schermo non sarà visualizzato alcun carattere. È possibile passare tutti i caratteri, inclusi i caratteri di controllo, tranne CTRL BREAK, che viene usato per interrompere l'esecuzione della funzione INPUT$. Per rispondere agli input da tastiera non è necessario premere il tasto ENTER.
La funzione INPUT$ vi offre la possibilità di leggere dalla tastiera caratteri significativi per l'editing, come BACKSPACE, che corrisponde al codice ASCII 8. Se volete leggere questi caratteri speciali, potete usare INPUT$ o INKEY$, ma non INPUT o LINE INPUT.
Per file di comunicazioni la funzione INPUT$ è preferibile rispetto alle funzioni INPUT$ e LINE INPUT$, dato che in questo caso tutti i caratteri ASCII hanno significato.

## **INSTR** Funzione

Cerca la prima occorrenza di *stringa2* in *stringa1* e restituisce la posizione in cui viene trovata l'uguaglianza. Il parametro opzionale *start* assegna la posizione di partenza per la ricerca in *stringa1*.

*Sintassi:*
*varnum* = INSTR ([*start*,] *stringa1*,*stringa2*)

*Esempi:*

```
PRINT INSTR (A$,B$)
PRINT INSTR (5,X$,Z$)
```

Il parametro *start* deve avere valore compreso tra 0 e 255; *stringa1* e *stringa2* possono essere variabili, espressioni, o costanti, ma sempre di tipo stringa.
Se *start* è posto maggiore della lunghezza di *stringa1*, allora o *stringa2* è la stringa nulla o non può essere trovata. Se *stringa2* è nulla, INSTR restituisce il valore di *start* (o il valore 1 se *start* non è stato specificato). Se *start* è al di fuori dell'intervallo permesso, verrà visualizzato il messaggio d'errore ILLEGAL FUNCTION CALL.

## **INT** Funzione

Restituisce il più grande numero intero che sia minore o uguale a *esprnum*. Per i numeri positivi equivale alla parte intera del numero.

*Sintassi:*
*varnum*= INT(*esprnum*)

*Esempio:*

```
PRINT INT (45.67)
```

Si noti che INT(−5.8)= −6. Vedere FIX e CINT, poiché entrambe restituiscono un valore intero.

## **KEY** (ALT K) Istruzione

Assegna o visualizza la funzione dei tasti definibili.

*Sintassi:*
KEY *n*,"*stringa*"
KEY LIST
KEY ON
KEY OFF

*Esempio:*

```
KEY 3, "NEW"+CHR$(13)
```

Il parametro *n* è il numero del tasto funzione nell'intervallo da 1 a 10 per i normali tasti funzione, e da 15 a 20 per i tasti speciali per le routine di intercettamento; *stringa* è un'espressione di tipo stringa che sarà assegnata al tasto (ricordatevi di racchiudere la costante *stringa* tra virgolette). Il precedente esempio assegnerà al tasto F3 il comando NEW←.
L'istruzione KEY permette ai tasti funzione di essere designati come tasti definibili dall'utente. Cioè voi potete assegnare ad ogni tasto funzione il compito di scrivere automaticamente una certa sequenza di caratteri. Ad ognuno dei 10 tasti funzione può essere assegnata una stringa di non più di 15 caratteri.
Inizialmente ai tasti funzione sono assegnati i seguenti valori:

| | | | |
|---|---|---|---|
| F1 | LIST | F2 | RUN← |
| F3 | LOAD" | F4 | SAVE" |
| F5 | CONT← | F6 | ,"LPT1:"← |
| F7 | TRON← | F8 | TROFF← |
| F9 | KEY | F10 | SCREEN 0,0,0← |

La freccia (←) indica ENTER.
L'istruzione KEY ON consente la visualizzazione dei valori dei tasti funzione sulla 25-esima riga dello schermo; se la larghezza è 40 colonne, vengono visualizzati 5 dei 10 tasti funzione, se è 80 colonne vengono visualizzati tutti e 10. In entrambi i casi, comunque, vengono visualizzati solo i primi 6 caratteri di ogni valore. ON è lo stato di default per la visualizzazione dei tasti funzione.
L'istruzione KEY OFF cancella dalla 25-esima riga le informazioni sui tasti funzione, rendendo la linea libera per il programma, senza disabilitare i tasti funzione stessi.
L'istruzione KEY LIST visualizza per intero sullo schermo l'elenco dei valori di tutti i 10 tasti funzione disponibili.
L'istruzione KEY *n,"stringa"* assegna il valore di *stringa* al tasto funzione specificato (da 1 a 10). *stringa* può essere lunga non più di 15 caratteri; anche se la lunghezza è maggiore vengono assegnati solo i primi 15. Assegnando la stringa nulla (cioè di lunghezza zero) si disabilita il tasto funzione come tasto definibile.
Se il valore assegnato a *n* non appartiene all'intervallo da 1 a 10, si incorre nell'errore ILLEGAL FUNCTION CALL e viene mantenuta la precedente stringa di assegnamento.
Quando viene premuto un tasto funzione definibile, la funzione INKEY$ restituisce, ogni volta che viene chiamata, un carattere della stringa del

tasto funzione stesso; se questo è disabilitato, INKEY$ restituisce una stringa di due caratteri: il primo carattere è il numero binario zero, il secondo è il codice di scansione del tasto.
Dopo aver disattivato la visualizzazione del tasto funzione con l'istruzione KEY OFF, potete usare LOCATE 25,1 seguita da PRINT per visualizzare quel che volete nell'ultima riga dello schermo. L'informazione sulla linea 25 non può essere fatta scorrere, mentre ciò è possibile per le linee dalla 1 alla 24.
Vi sono modi di intercettamento definibili dall'utente, che permettono di intercettare tutti i tasti di SHIFT. Questi tasti sono definiti con l'istruzione:

KEY *n*,CHR$(*stato-shift*)+CHR$(*codice di scansione*)

dove *n* è un numero compreso nell'intervallo da 15 a 20, *stato-shift* è il valore numerico che corrisponde al valore esadecimale del tasto di shift che deve essere premuto, e il *codice di scansione*, con valore tra 1 e 83, identifica il tasto da intercettare. (Vedi Figura 6.3, pag. 213, per la tabella dei codici di scansione).
I valori esadecimali per *stato-shift* sono:

| | |
|---|---|
| CAPS LOCK | &H40 |
| NUM LOCK | &H20 |
| ALT | &H08 |
| CTRL | &H04 |
| SHIFT | &H01, &H02 e &H03 |

Naturalmente, *stato-shift* per i tasti non shiftati vale &H00.
I due tasti SHIFT, quello a destra e quello a sinistra, sono funzionalmente equivalenti e perciò uno qualsiasi tra i valori &H01, &H02 e &H03 si riferisce ad entrambi i tasti. Potete anche combinare tra loro diversi stati di shift, come i due tasti CTRL e ALT, ad esempio, (&H04+&H08 = &H0C).
I valori di shift devono sempre essere dati in notazione esadecimale.
La routine di intercettamento dei tasti, quando utilizzata, funziona così:

1. Dapprima viene elaborato il carattere CTRL PRTSC, che attiva la stampante. Anche se è stato definito come tasto di intercettamento, CTRL PRTSC può ancora essere utilizzato per ottenere una copia stampata di ciò che appare sullo schermo.
2. Successivamente vengono elaborati i tasti funzione da F1 a F10 e i tasti di controllo del cursore (su, giù, destra, sinistra), cioè i tasti da 1 a 14. Definire i codici di scansione da 59 a 68, oltre a 72, 75, 77 o 80 come tasti di intercettamento non ha alcun effetto, perché questi si considerano predefiniti.
3. Infine, vengono elaborati i tasti da voi definiti con valori da 15 a 20.

NOTA:

1. I tasti intercettati non passano nel buffer della tastiera.
2. Fate attenzione nell'utilizzare come tasti di intercettamento le combinazioni CTRL BREAK e CTRL ALT DEL, perché, a meno che non inseriate un controllo nella routine di intercettamento, per interrompere il programma non vi rimane altra soluzione che spegnere il computer.

## KEY (n) Istruzione

Attiva e disattiva l'intercettamento del tasto specificato in un programma BASIC (vedi ON KEY(*n*)). Utilizzabile solo in Advanced BASIC ed in Compiled BASIC.

*Sintassi:*

KEY(*n*) ON
KEY(*n*) OFF
KEY(*n*) STOP

*Esempio:*

```
KEY (x+1) ON
```

Il parametro *n*, che deve appartenere all'intervallo di valori da 1 a 20, indica il tasto da intercettare nel seguente modo:

| | |
|---|---|
| 1-10 | tasti funzione da F1 a F10 |
| 11 | cursore su (↑) |
| 12 | cursore a sinistra (←) |
| 13 | cursore a destra (→) |
| 14 | cursore giù (↓) |
| 15-20 | tasti definiti nella forma KEY*n*, CHR$(*stato-shift*)+CHR$(*codice di scansione*) |

L'istruzione KEY(*n*) ON deve essere eseguita per attivare l'intercettamento di tasti funzione o l'attività del tasto di controllo del cursore. Dopo l'esecuzione, se nell'istruzione ON KEY(*n*) era stato specificato un numero di linea diverso da zero, ogni volta che il BASIC inizia una nuova istruzione andrà anche a controllare se il tasto specificato era stato premuto; se sì, eseguirà l'istruzione GOSUB che lo porta al numero di linea specificato nell'istruzione ON KEY(*n*).
Se invece l'istruzione KEY(*n*) è OFF, non viene predisposto nessun inter-

cettamento, e se il tasto viene premuto, l'evento non viene memorizzato. Dopo l'esecuzione di un'istruzione KEY(*n*) STOP, non verrà predisposto alcun intercettamento. Comunque, se premete il tasto specificato la vostra azione verrà memorizzata in modo che sia effettuato un intercettamento immediato alla successiva esecuzione di KEY(*n*) ON.
L'istruzione KEY(*n*) ON non ha effetto alcuno sui valori dei tasti funzione definibili visualizzati sul fondo dello schermo.
Se usate un'istruzione KEY(*n*) in Disk BASIC, otterrete in risposta il messaggio d'errore SYNTAX ERROR. Vedere KEY per ulteriori spiegazioni sull'uso dell'istruzione KEY senza *n*.

## **KILL** Comando

Cancella i file dal disco. Il comando KILL in BASIC è simile al comando ERASE del DOS e non è utilizzabile in Compiled BASIC.

*Sintassi:*
KILL "*specfile*"

*Esempio:*

```
KILL "B:PROVA.BAS"
```

Il nome del dispositivo specificato in *specfile* deve essere quello di un disco e, se viene omesso, il DOS usa il drive di default; inoltre *specfile* può contenere un cammino. Il comando KILL può essere usato per tutti i file su disco. Il nome deve includere l'estensione, qualora esista; per esempio, se salvate un programma BASIC usando il comando:

```
SAVE "TEST"
```

il BASIC stesso supplirà la mancanza dell'estensione, aggiungendo .BAS. Ciò non avviene con il comando KILL: infatti se in seguito volete cancellare quel programma TEST, dovrete dare il comando

```
KILL "TEST.BAS"
```

e non il comando

```
KILL "TEST"
```

Se date il comando KILL per un file che in quel momento è aperto, sarà visualizzato il messaggio d'errore FILE ALREADY OPEN. Il comando KILL può essere usato solo per cancellare dei file, mentre per rimuovere dei directory deve essere usato il comando RMDIR.

## **LEFT$** Funzione

Restituisce gli *esprnum* caratteri più a sinistra in *stringa.*

*Sintassi:*
*varstr* = LEFT$(*stringa,esprnum*)

*Esempio:*

```
X$=LEFT$(Y$,C)
```

Il parametro *esprnum*, che deve appartenere all'intervallo da 0 a 255, specifica il numero di caratteri che devono essere contenuti nel risultato. Nel caso in cui *esprnum* sia maggiore della lunghezza della stringa, verrà restituita l'intera stringa; se *esprnum* è uguale a zero, viene restituita la stringa nulla.

## **LEN** Funzione

Restituisce il numero di caratteri in una stringa.

*Sintassi:*
*varnum* = LEN("*stringa*")

*Esempio:*

```
Q=LEN("FIAT S.p.A.")
```

I caratteri non stampabili e gli spazi bianchi vengono inclusi nel conteggio del numero di caratteri.

## **LET** Istruzione

Assegna ad una variabile il valore di un'espressione.

*Sintassi:*
[LET] *variabile*=*espressione*

*Esempi:*

```
LET A=B
Q(X)=Q(X+1)
```

Il parametro *variabile* è il nome di una variabile o di un elemento di un array che deve ricevere un valore; può essere una variabile o un elemento di un array sia di tipo stringa che numerico. Il parametro *espressione* è l'espressione il cui valore sarà assegnato a *variabile*; il tipo di espressione (numerico o stringa) deve essere uguale al tipo della variabile, altrimenti si incorre nell'errore TYPE MISMATCH. Si noti che la parola LET è opzionale, cioè che il segno = è sufficiente per l'assegnamento del valore di un'espressione ad una variabile.

## LINE Istruzione

Disegna una linea o un rettangolo sullo schermo.

*Sintassi:*
LINE[(*x1,y1*)] – (*x2,y2*)[,[*colore*] [,B[F]] [,*stile*]]

*Esempi:*

```
LINE (0,80)-(100,100),2,BF
LINE -(90,0)
LINE (300,0)-(200,50)
```

I parametri (*x1,y1*) e (*x2,y2*) sono le coordinate in forma assoluta o relativa; *colore* è il numero del colore, in un intervallo da 0 a 3; in media risoluzione seleziona il colore dalla tavolozza corrente come definita dall'istruzione COLOR, ed il colore di fondo è 0. Il colore del testo di default è il numero 3. In alta risoluzione, invece, il numero 0 indica il nero, mentre il colore di default è il bianco cui corrisponde il numero 1. Il parametro opzionale *stile* definisce una maschera a 16 bit, che viene usata per disegnare punti sullo schermo.
La forma più semplice di LINE è:

LINE – (*x2,y2*)

Quest'istruzione disegnerà una linea a partire dall'ultimo punto di riferimento fino al punto (*x2,y2*) nel colore del testo.
Possiamo anche definire un punto di inizio del disegno.

```
LINE (0,0)-(319,199)          diagonale lungo lo schermo
LINE (0,100)-(319,100)        barra attraverso lo schermo
```

Possiamo anche indicare il colore con cui disegnare nel modo seguente:

```
LINE (10,10)-(20,20),2        disegna col colore 2
```

```
10 SCREEN 1,0,0,0:CLS     disegna linee casuali in un colore casuale
20 LINE -(RND*319,RND*199),RND*4
30 GOTO 20
```

```
10 FOR X=0 TO 329             disegna linee alternate
20 LINE (X,0)-(X,199),X AND 1
30 NEXT
```

L'argomento successivo della funzione LINE è B (*box* = rettangolo) o BF (*filled box* = rettangolo pieno). Possiamo togliere *colore* e includere l'argomento finale,

```
LINE (0,0)-(100,100),,B       rettangolo nel colore di testo
```

o anche includere il colore,

```
LINE (0,0)-(100,100),2,BF  rettangolo pieno nel colore 2
```

Il carattere B dice al BASIC di disegnare un rettangolo con i punti ($x1,y1$) e ($x2,y2$) come angoli opposti, e questo evita di dover dare i quattro comandi LINE che hanno lo stesso risultato:

```
LINE (X1,Y1)-(X2,Y1)
LINE (X1,Y1)-(X1,Y2)
LINE (X2,Y1)-(X2,Y2)
LINE (X1,Y2)-(X2,Y2)
```

I caratteri BF dicono al computer di disegnare lo stesso rettangolo, ma di riempire i punti interni con il colore selezionato.
L'opzione *stile* dell'istruzione LINE definisce una maschera in cui il corrente bit in *stile* viene usato per disegnare un punto della linea dello schermo. Se il bit è zero, il punto non viene colorato, se invece il bit è uguale ad uno, viene colorato il corrispondente punto della linea. Dopo ogni punto, viene utilizzata in *stile* la posizione del bit successivo e, quando è stata selezionata l'ultima, l'istruzione LINE si volgerà indietro e ricomincerà dalla posizione del primo bit.
Si noti che un bit zero salta sopra al punto dello schermo ma non lo can-

cella; così potreste disegnare una linea di fondo prima di una linea con l'opzione *stile*, per ottenere il colore di fondo voluto anche nella linea. Per meglio vedere la potenza dell'opzione *stile*, potete disegnare una linea punteggiata attraverso lo schermo colorando un punto sì ed uno no. Poiché *stile* ha la lunghezza di 16 bit, la maschera per la linea tratteggiata sarà:

```
1010101010101010
```

Questo è il valore esadecimale AAAA: così *stile* avrà il valore &HAAAA. L'opzione *stile* può essere usata per disegnare solo linee normali e rettangoli: utilizzando l'opzione BF con l'opzione *stile* si incorre nell'errore SYNTAX ERROR.
Se all'interno dell'istruzione LINE usate delle coordinate al di fuori dell'intervallo consentito, la linea verrà troncata: la parte di immagine che è al di fuori dell'intervallo delle coordinate verrà tagliata ai bordi dello schermo.
L'ultimo punto di riferimento dopo l'istruzione LINE è il punto (*x2,y2*). Se per le seconde coordinate usate la forma relativa, lo fate in relazione alle prime coordinate; ad esempio:

```
LINE (100,100)-STEP(10,-20)
```

disegnerà una linea da (100,100) a (110,80).

## **LINE INPUT** Istruzione

Legge dalla tastiera un'intera linea (fino a 254 caratteri) e la copia in una variabile stringa, ignorando i delimitatori.

*Sintassi:*
LINE INPUT [;] ["*stringa prompt*";] *varstr*

*Esempio:*

```
LINE INPUT "Introdurre la cifra"; C$
```

Il parametro *stringa prompt* è una stringa che viene visualizzata sullo schermo prima che l'input venga accettato. Il punto interrogativo non viene visualizzato a meno che non faccia parte di *stringaprompt*. Il parametro *varstr* è il nome di una variabile stringa o di un elemento di un array cui verrà assegnata la linea. Tutti i caratteri inviati in input a partire dall'ultimo carattere di *stringa prompt* fino al punto in cui viene premu-

to il tasto ENTER vengono assegnati alla variabile *varstr*; si tenga presente che vengono ignorati gli spazi bianchi che terminano la stringa.
In Disk BASIC ed in Advanced BASIC, se l'input è immediatamente seguito da un punto e virgola, premendo il tasto ENTER per terminare la linea di input non si produrrà sullo schermo il ritorno a capo (CR/LF), cioè il cursore rimane sulla stessa linea della vostra risposta.
Potete uscire da LINE INPUT premendo i tasti CTRL, BREAK, ed il BASIC ritorna al livello comandi e visualizza Ok. Potete poi anche inviare il comando CONT per riprendere l'esecuzione dell'istruzione LINE INPUT.

## **LINE INPUT#** Istruzione

Legge un'intera linea (fino a 254 caratteri), ignorando i delimitatori, da un file sequenziale e la copia in una variabile stringa.

*Sintassi:*
LINE INPUT# *numfile,varstr*

*Esempio:*

```
LINE INPUT #2, A$
```

Il parametro *numfile* è il numero con il quale il file era stato aperto; *varstr* è il nome di una variabile stringa o un elemento dell'array cui la linea verrà assegnata.
L'istruzione LINE INPUT# legge tutti i caratteri del file sequenziale fino al carattere CR; a questo punto oltrepassa il carattere CR/LF (a capo), e la successiva istruzione LINE INPUT# legge tutti i caratteri fino al successivo CR. (Se viene trovato un carattere LF/CR viene conservato, cioè viene passato come carattere del file).
L'istruzione LINE INPUT# è utile soprattutto se ogni linea di un file è stata suddivisa in vari campi, o se un programma BASIC salvato in modo ASCII deve essere letto come dato di un altro programma.

## **LIST** Comando

Invia allo schermo o ad altro dispositivo il listato del programma corrente in memoria. Non è utilizzabile in Compiled BASIC.

Sintassi:
LIST [*linea1*] [–[*linea2*]] [,"*specfile*"]

*Esempi:*

```
LIST
LIST 100
LIST 200
LIST, "A:TRIAL"
```

I parametri *linea1* e *linea2* sono dei numeri di linea compresi nell'intervallo da 0 a 65529; *linea1* è la prima linea da visualizzare e *linea2* è l'ultima. Può essere usato un punto per entrambi i numeri di linea ad indicare la linea corrente. Non è necessario che le linee esistano. Se viene omesso il parametro *specfile*, le linee indicate vengono visualizzate sullo schermo. Ogni listato che venga inviato sia sullo schermo che sulla stampante può essere interrotto premendo i tasti CTRL BREAK, mentre CTRL NUM LOCK lo sospenderà solamente; si potrà ricominciare premendo un tasto qualsiasi. Se viene omesso l'intervallo di linee da listare, l'operazione viene compiuta sull'intero programma. Quando invece viene usata la lineetta, sono possibili le seguenti tre opzioni:

— Se viene passata solo *linea1*, vengono listate solo quella linea e tutte quelle con numero di linea maggiore.
— Se viene passata solo *linea2*, vengono listate tutte le linee del programma dall'inizio fino a quella indicata
— Se vengono passate sia *linea1* che *linea2*, vengono listate tutte le linee con numero di linea compreso tra quei due.

Se *linea1* (o *linea2*) non esiste, verrà usata la successiva linea esistente. Quando listate in un file su disco, la parte specificata del programma viene salvata in formato ASCII e potrà essere usata più avanti per mezzo dell'istruzione MERGE.

## **LLIST** Comando

Lista tutto o una parte del programma corrente in memoria sulla stampante (LPT1:). Non è utilizzabile in Compiled BASIC.

*Sintassi:*
LLIST [*linea1*] [–[*linea2*]]

*Esempi:*

```
LLIST
LLIST.-
LLIST 100-1000
```

Gli intervalli dei numeri di linea hanno la stessa funzione di quelli di LIST. Il BASIC ritorna sempre al livello comando dopo aver eseguito un'istruzione LLIST.

## **LOAD** Comando

Carica un programma da uno specifico dispositivo all'interno della memoria, con la possibiltà di avviarne l'esecuzione. Non è utilizzabile in Compiled BASIC.

*Sintassi:*
LOAD "*specfile*"[,R]

*Esempi:*

```
LOAD "A:TRIG"
LOAD "B:PROG",R
```

L'istruzione LOAD chiude tutti i file aperti e cancella tutte le variabili e le linee del programma residente in memoria, prima di caricare il programma specificato. Se l'opzione R viene omessa, il BASIC ritorna al modo diretto dopo aver caricato il programma.
Comunque, se con il comando LOAD viene usata l'opzione R, il programma andrà in esecuzione subito dopo essere stato caricato. In questo caso tutti i file di dati aperti vengono mantenuti tali. Così LOAD con l'opzione R può essere usata per concatenare più programmi (o segmenti dello stesso programma). Le informazioni necessarie possono essere passate tra programmi usando file di dati.
L'istruzione LOAD "*specfile*",R è equivalente a RUN "*specfile*". Quando viene omesso il nome del dispositivo da cui prelevare il file, viene usato il drive di default del DOS. L'estensione .BAS viene aggiunta al nome del file qualora non venga passata l'estensione ed il nome del file sia di non più di 8 caratteri.

## **LOC** Funzione

Restituisce la posizione corrente all'interno del file.

*Sintassi:*
*varnum* = LOC(*numfile*)

*Esempio:*

```
IF LOC(3)=0 THEN 1000
```

Il parametro *numfile* è il numero usato per aprire il file. Con i file ad accesso diretto LOC restituisce il numero identificativo dell'ultimo record letto o scritto.
Con i file sequenziali, invece, LOC restituisce il numero di record letti o scritti finché il file era rimasto aperto. (Un record è un blocco di dati di 128 byte di lunghezza). Quando un file viene aperto per degli input sequenziali, il BASIC legge il primo settore del file, cosicché LOC restituisce un 1 anche prima di ogni input dal file.
Per un file di comunicazioni, la funzione LOC restituisce il numero di caratteri del buffer di input che sono in attesa di essere letti. La dimensione di default per il buffer di input è 256 caratteri, ma può essere cambiata utilizzando l'opzione /C: nel comando BASIC; se nel buffer vi sono più di 255 caratteri, il valore restituito da LOC è 255. Poiché una stringa è limitata a 255 caratteri, questa limitazione pratica vi solleva dall'incombenza di dover controllare la lunghezza della stringa prima di andarvi a leggere dei dati. Se nel buffer restano meno di 255 caratteri, LOC restituisce il numero effettivo.

## **LOCATE** (ALT L) Istruzione

Posiziona il cursore sullo schermo attivo; i parametri opzionali accendono e spengono il cursore e ne definiscono le dimensioni.

*Sintassi:*
LOCATE [*riga*] [,[*colonna*][,[*cursore*][,[*start*] [,*stop*]]]]

*Esempi:*

```
LOCATE 1,1
LOCATE ,,,7
LOCATE 5,1,1,0,7
```

Tutti i parametri sono delle espressioni numeriche. Il primo, *riga*, che deve appartenere all'intervallo da 1 a 25, indica il numero di riga dello schermo in cui volete posizionare il cursore; *colonna*, che deve avere valori nell'intervallo da 1 a 40 o da 1 a 80, a seconda delle dimensioni dello schermo, indica invece il numero di colonna. Il parametro *cursore* sta ad indicare se si desidera che il cursore resti visibile oppure scompaia durante l'esecuzione del programma: il valore 0 (di default) indica acceso, 1 indica spento. Infine, *start*, con valori tra 0 e 31, è la riga di punti di par-

tenza del cursore, mentre *stop*, sempre tra 0 e 31, è la riga di arresto del cursore.
Tutti questi parametri non sono utilizzabili in modo Grafico. I parametri *start* e *stop* vi permettono di dare al cursore le dimensioni che volete, dato che indicano la riga di inizio e di arresto. Le righe sono numerate da 0 alla posizione massima del carattere. La linea inferiore è 7 se possedete la scheda Colore/Grafica, oppure 13 se possedete l'interfaccia per video monocromatico e stampante parallela. Quando *stop* viene omesso e *start* viene passato, *stop* assume il valore di *start*; se *start* è maggiore di *stop*, vi troverete un cursore in due parti; il cursore può essere reso invisibile assegnando a *start* un valore maggiore della riga inferiore, con un valore del parametro *stop* minore o uguale ancora della riga inferiore. L'istruzione LOCATE,,1,7,7 riporta il cursore alla sua forma originaria.
Dopo l'esecuzione dell'istruzione LOCATE, le istruzioni di I/O cominciano a porre sullo schermo dei caratteri alla locazione specificata. Normalmente il BASIC non scriverà sulla 25-esima riga, comunque potete disabilitare i tasti funzione definibili usando KEY OFF, quindi dare l'istruzione LOCATE 25,1: PRINT... per poter scrivere anche sulla linea 25.
Naturalmente ognuno dei parametri può essere omesso, con la limitazione che se viene omesso *start*, deve essere omesso anche *stop*. Per i valori non assegnati viene assunto il valore corrente. Ogni volta che si cerca di passare un valore al di fuori dell'intervallo consentito, si provoca la visualizzazione del messaggio d'errore ILLEGAL FUNCTION CALL, e vengono mantenuti i valori precedenti.

## **LOF** Funzione

Restituisce il numero di byte presenti in un file, cioè la lunghezza del file.

*Sintassi:*

*varnum* = LOF(*numfile*)

*Esempio:*

```
PRINT LOF(3)
```

Il parametro *numfile* è il numero usato per aprire il file. Per i file creati dal BASIC, LOF restituirà il numero effettivo di byte presenti nel file.
Per file per comunicazioni, LOF restituisce la quantità di spazio libero all'interno del buffer di input, cioè *dimensione*-LOC(*numfile*), ove il para-

metro *dimensione* rappresenta la dimensione del buffer di comunicazione, che per default è 256, ma che può essere cambiato con l'opzione /C: nei comandi BASIC. LOF può essere usata per controllare quando il buffer di input è pieno.

## LOG Funzione

Restituisce il logaritmo naturale di *esprnum*.

*Sintassi:*
*varnum* = LOG(*esprnum*)

*Esempio:*

```
PRINT LOG(37)
```

Il logaritmo naturale, che può essere calcolato solo per *esprnum* maggiore di zero, è il logaritmo in base *e*. Il calcolo di LOG viene fatto in precisione semplice a meno che il BASIC non sia stato avviato con l'opzione /D.

## LPOS Funzione

Restituisce il valore della posizione corrente nel buffer della stampante della testina di stampa.

*Sintassi:*
*varnum* = LPOS(*esprnum*)

*Esempio:*

```
IF LPOS(0)>60 THEN 100
```

Il parametro *esprnum* indica quale stampante deve essere controllata, nel modo seguente:

0 o 1 LPT1:
2 LPT2:
3 LPT3:

La funzione LPOS non restituisce necessariamente la posizione fisica della testina sulla stampante, ma solo la posizione virtuale nel buffer.

## **LPRINT e LPRINT USING** Istruzione

Invia dati in stampa alla stampante (LPT1:).

*Sintassi:*
LPRINT [*lista di espressioni*] [;]
LPRINT USING "*stringa*"; *lista di espressioni* [;]

*Esempi:*

```
LPRINT QR$,B
LPRINT USING "####.##";B
```

Il parametro *lista di espressioni* è una lista di espressioni numeriche e a stringa che devono essere stampate; queste espressioni devono essere separate da virgole o da punti e virgola. Il parametro *stringa* invece è una costante o una variabile stringa che identifica il formato da usare per la stampa. Queste istruzioni funzionano come le istruzioni PRINT e PRINT USING eccettuato il fatto che in questo caso l'output avviene su stampante e non su video (vedi PRINT e PRINT USING).
L'istruzione LPRINT assume una stampante con lunghezza di riga di 80 caratteri; cioè il BASIC inserisce automaticamente un CR/LF dopo aver stampato 80 caratteri. Questo comporta il ritorno a capo di due righe quando stampate esattamente 80 caratteri, a meno che non terminiate l'istruzione con un punto e virgola. Potete inoltre cambiare la larghezza con un'istruzione WIDTH"LPT1:".
La stampa avviene in maniera asincrona rispetto all'elaborazione. Se date un salto pagina (*form feed* = LPRINT CHR$(12);) seguito da un'altra istruzione LPRINT e la stampante impiega più di 10 secondi per eseguirlo, potreste ottenere il messaggio d'errore DEVICE TIMEOUT. Per risolvere il problema basterà fare:

```
1 ON ERROR GOTO 65000
   .
   .
   .
65000 IF ERR = 24 THEN RESUME '24=timeout
```

Potete controllare ERL per essere sicuri che il *timeout* (fuori tempo massimo) sia causato dall'istruzione LPRINT.

## **LSET e RSET** Istruzione

Muove dati all'interno del buffer di un file ad accesso diretto.

*Sintassi:*
LSET *varstr* = *stringa*
RSET *varstr* = *stringa*

*Esempio:*

```
LSET F$=I$
RSET D$=B$
```

Il parametro *varstr* è il nome di una variabile definita nell'istruzione FIELD; *stringa* è un'espressione di tipo stringa per le informazioni da collocare nel campo identificato da *varstr*.
Se *stringa* richiede meno byte di quanti ne erano specificati per *varstr* nell'istruzione FIELD, l'istruzione LSET giustifica a sinistra la stringa nel campo e RSET la giustifica a destra (per riempire le posizioni aggiunte vengono inseriti degli spazi bianchi). Se *stringa* è più lunga di *varstr*, i caratteri vengono tagliati a destra.

NOTA: le istruzioni LSET e RSET possono anche essere usate con una variabile stringa che non era stata definita in un'istruzione FIELD per giustificare a destra e a sinistra una stringa in un dato campo. Per esempio, le linee di programma

```
110 A$=SPACE$(20)
120 RSET A$=N$
```

giustificano a destra la stringa N$ in un campo di 20 caratteri. Questo può essere utile per formattare degli output da stampare.

## **MERGE** Comando

Fonde le linee di un file di programma in formato ASCII nel programma corrente in memoria. Non è utilizzabile in Compiled BASIC.

*Sintassi:*
MERGE "*specfile*"

*Esempio:*

```
MERGE "A:NUMERO"
```

Il file indicato viene cercato nel dispositivo presente in *specfile* e, se viene trovato, le linee del file nel dispositivo vengono fuse con le linee in

memoria. Se qualche linea del file da fondere ha lo stesso numero di linea delle linee del programma in memoria, le linee del file sostituiranno le corrispondenti linee in memoria.
Dopo l'esecuzione del programma MERGE, il programma fuso risiede in memoria ed il BASIC ritorna al livello comandi. Se viene omesso il nome del dispositivo, viene preso dal DOS il drive di default.
Se il programma da fondere non era stato salvato in formato ASCII (usando cioè l'opzione A nel comando SAVE), si incorrerà nell'errore BAD FILE MODE, ed il programma in memoria resterà immutato.

## **MID$** Funzione

Restituisce la parte richiesta di una data stringa.

*Sintassi:*
*varstr* = MID$(*stringa, start* [,*lunghezza*])

*Esempio:*

```
NA$=MID$(ID$,3,20)
```

I parametri *start* e *lunghezza* sono delle espressioni intere nell'intervallo rispettivamente da 1 a 255 e da 0 a 255. La funzione restituisce una stringa di *lunghezza* caratteri presa da *stringa* cominciando dal carattere *start*. Se *lunghezza* viene omessa, o se a destra del carattere *start* vi sono più di *lunghezza* caratteri, vengono restituiti tutti i caratteri più a destra del carattere *start*. Se *lunghezza* = 0, o se *start* è maggiore di LEN(*stringa*), la funzione MID$ restituisce la stringa nulla.

## **MID$** Istruzione

Rimpiazza una parte di stringa con un'altra stringa.

*Sintassi:*
MID$(*varstr, start* [,*lunghezza*]) = "*stringa*"

*Esempio:*

```
MID$(A$,13,1)="P"
```

Il parametro *start* è un'espressione intera nell'intervallo da 1 a 255; *lunghezza* è un'espressione intera tra 0 e 255. I caratteri in *varstr* a partire dalla posizione *start*, sono rimpiazzati dai caratteri in *stringa*. Il parametro opzionale *lunghezza* si riferisce al numero di caratteri di *stringa* che devono essere usati per il rimpiazzo e, se viene omesso, è usata l'intera *stringa*.

Comunque, indipendentemente dal fatto che *lunghezza* venga omessa o meno, la lunghezza di *varstr* non cambia. Per esempio, se *varstr* è di 4 caratteri e *stringa* di 5, dopo il rimpiazzo, *varstr* conterrà solo i primi 4 caratteri di stringa.

NOTA: se sia *start* che *lunghezza* eccedono l'intervallo di valori consentito, verrà visualizzato il messaggio d'errore ILLEGAL FUNCTION CALL.

## **MKDIR** Comando

Crea un nuovo directory nel disco specificato. Utilizzabile solo in Advanced BASIC ed in Disk BASIC.

*Sintassi:*
MKDIR "*nomedir*"

*Esempio:*

```
MKDIR "VENDITE"
```

Il parametro *nomedir* è un'espressione stringa che identifica il nuovo directory da creare, e non deve essere lunga più di 63 caratteri. Vedere il Capitolo 3 per ulteriori informazioni sulle strutture ad albero, ed i comandi CHDIR e RMDIR per cambiare e cancellare cataloghi.

## **MKI$, MKS$, MKD$** Funzioni

Convertono valori di tipo numerico in valori di tipo stringa.

*Sintassi:*
*varstr* = MKI$(*espressione intera*)
*varstr* = MKS$(*espressione in precisione semplice*)
*varstr* = MKD$(*espressione in doppia precisione*)

*Esempi:*

```
D$=MKI$(Q%)
R$=MKS$(D!)
M$=MKD$(S#)
```

Ogni valore numerico che viene posto in un buffer di un file ad accesso casuale con un'istruzione LSET e RSET deve essere convertito in una stringa. MKI$ converte un intero di una stringa di 2 byte. MKS$ converte un numero in precisione semplice in una stringa di 4 byte. MKD$ conver-

te un numero in doppia precisione in una stringa di 8 byte.
Queste funzioni differiscono da STR$ in quanto non cambiano effettivamente i byte del dato, ma proprio il modo in cui il BASIC interpreta questi byte. Vedere anche CVI, CVS, CVD.

## **MOTOR** (ALT M) Istruzione

Accende e spegne da programma il registratore a cassette. Non è utilizzabile in Compiled BASIC o sul PC/XT.

*Sintassi:*
MOTOR [*esprnum*]

*Esempio:*

```
MOTOR 1
```

Se il parametro *esprnum* è diverso da 0, il registratore viene acceso, se è uguale a 0 viene spento. Se *esprnum* è omesso, viene commutato lo stato del registratore, cioè se è acceso, viene spento, e viceversa.

## **NAME** Comando

Cambia il nome di un file su disco; è simile al comando RENAME del DOS.

*Sintassi:*
NAME "*specfile*" AS "*nomefile*"

*Esempio:*

```
NAME "A:ATTO.BAS" AS "COMM.BAS"
```

Il file specificato in *specfile* deve essere presente sul disco, mentre non deve esserlo il file *nomefile*, in caso contrario si incorrerebbe in un errore. Se viene omesso il nome del dispositivo in *specfile*, viene usato il drive di default del DOS; si noti che l'estensione del file non è posta per default uguale a .BAS. Dopo l'esecuzione di un comando NAME, il file con il nome nuovo rimane nella stessa area del disco occupata precedentemente.

## **NEW** Comando

Cancella dalla memoria il programma corrente e tutte le variabili. Non è utilizzabile in Compiled BASIC.

*Sintassi:*
NEW

*Esempio:*

```
NEW
```

Questo comando viene solitamente usato per liberare spazio di memoria prima di caricare un nuovo programma. Subito dopo la sua esecuzione il BASIC ritorna al livello comandi. Inoltre, tutti i file aperti vengono chiusi e viene disabilitata la traccia del programma (vedi comando TRON e TROFF).

## **OCT$** Funzione

Restituisce una stringa che rappresenta il valore ottale di un argomento decimale.

*Sintassi:*
*varstr* = OCT$(*esprnum*)

*Esempio:*

```
PRINT OCT$(24)
```

Se il parametro *esprnum*, che deve appartenere all'intervallo da −32768 a 65535, è negativo, allora viene calcolato il complemento a due, cioè OCT$(*esprnum*) equivale a OCT$(*esprnum*−65536). Vedere HEX$ per la conversione esadecimale.

## **ON COM (n)** Istruzione

Assegna un numero di linea per il BASIC da cui far partire la routine di intercettamento relativa all'ingresso di un'informazione nel buffer di comunicazione. Valida solo in Advanced BASIC ed in Compiled BASIC.

*Sintassi:*
ON COM (*adattatore*) GOSUB *linea*

*Esempio:*

```
ON COM(1) GOSUB 4000
```

Il parametro *adattatore* è il numero dell'interfaccia di comunicazione, 1 o 2; *linea* è il numero di linea della routine di intercettamento: assegnandole zero si disabilita l'intercettamento dell'attività di comunicazione per l'interfaccia specificata.
Per attivare quest'istruzione per *adattatore* deve prima essere eseguita un'istruzione COM(*adattatore*) ON. Se, dopo l'istruzione COM(*adattatore*) ON, viene specificato in ON COM(*adattatore*) un numero di linea diverso da zero, ogni volta che il programma esegue una nuova istruzione il BASIC controlla se nella specificata interfaccia di comunicazione sia arrivato un qualche carattere: se è così, il BASIC esegue un'istruzione GOSUB alla linea specificata in *linea*.
Se invece viene eseguita un'istruzione COM(*adattatore*) OFF, non viene predisposto nessun intercettamento per l'interfaccia, ed inoltre, se vi sono attività di comunicazione, l'evento non viene memorizzato. Anche nel caso in cui venga eseguita un'istruzione COM(*adattatore*) STOP non vengono predisposti intercettamenti per l'interfaccia; comunque tutti i caratteri ricevuti vengono memorizzati, cosicché non appena viene eseguita un'istruzione COM(*adattatore*) ON, ha occorrenza un intercettamento. Quando si verifica un intercettamento, automaticamente viene eseguita un'istruzione COM(*adattatore*) STOP, cosicché non possono avvenire intercettamenti recursivi.
L'esecuzione dell'istruzione RETURN dalla routine di intercettamento provoca a sua volta l'esecuzione automatica dell'istruzione COM(*adattatore*) ON, a meno che non sia già stata eseguita un'istruzione COM(*adattatore*) OFF all'interno della routine stessa. Non possono avvenire eventi di intercettamento se il BASIC non sta eseguendo un programma. Quando viene intercettato un errore (risultato dell'istruzione ON ERROR), tutti gli intercettamenti vengono automaticamente disabilitati, inclusi ERROR, STRIG(*n*), PEN, COM(*n*), PLAY(*n*), TIMER e KEY(*n*).
Solitamente la routine di intercettamento di comunicazioni legge, prima di ritornare, un intero messaggio dalla linea di comunicazione. Non è raccomandabile che usiate l'intercettamento di comunicazione per messaggi di singoli caratteri poiché ad alta velocità di trasmissione il tempo speso dal sistema operativo per intercettare e leggere un singolo carattere può causare un *overflow* (straripamento) del buffer degli interrupt per comunicazioni.
Potete usare l'istruzione RETURN *linea* per tornare al programma BASIC ad un numero di linea fissato. Comunque questo ritorno non locale deve essere usato con cura, poiché un altro GOSUB, WHILE, FOR che sia attivo al momento dell'intercettamento, rimarrà attivo comunque.

## ON ERROR Istruzione

Abilita l'intercettamento di errori e specifica la prima linea della routine di manipolazione degli errori.

*Sintassi:*
ON ERROR GOTO *linea*

*Esempio:*

```
ON ERROR GOTO 100
```

Il parametro *linea* è il numero di linea della prima linea della routine di intercettamento degli errori: se non esiste, si incorre nell'errore UNDEFINED LINE NUMBER.
Per interrompere l'intercettamento degli errori, basta eseguire l'istruzione ON ERROR GOTO 0, gli errori successivi visualizzeranno un messaggio d'errore e interromperanno l'esecuzione. L'istruzione ON ERROR GOTO 0 all'interno della routine di intercettamento degli errori causa l'arresto del BASIC e la visualizzazione del messaggio d'errore relativo all'errore che ha causato l'intercettamento. Si consiglia di inserire in ogni subroutine di intercettamento un'istruzione ON ERROR GOTO 0 per tutti quegli errori per cui non esiste un'azione di recupero.

NOTA: se si incorre in un errore durante l'esecuzione di una subroutine di gestione degli errori, viene visualizzato il messaggio d'errore sullo schermo e l'esecuzione termina. L'intercettamento degli errori non avviene all'interno della routine di gestione degli errori. Per uscire dalla routine di intercettamento degli errori dovete usare l'istruzione RESUME.

## ON GOSUB e ON GOTO Istruzioni

Trasferiscono l'esecuzione ad uno dei numeri di linea specificati, secondo il valore dell'espressione.

*Sintassi:*
ON *espressione* GOTO *linea* [,*linea*]
ON *espressione* GOSUB *linea* [,*linea*]

*Esempio:*

```
ON N+1 GOSUB 100,200,300
ON Q GOTO 10,100
```

Il parametro *espressione* è un'espressione numerica arrotondata ad intero, quando necessario, variabile tra 0 e 255; *linea* è il numero di linea della linea di programma a cui volete saltare.
Il valore di *espressione* determina quale delle linee della lista verrà utilizzata per il salto: se il valore di *espressione* è 3, quindi, la destinazione del salto sarà la linea che occupa la terza posizione dell'elenco.
Nell'istruzione ON GOSUB, ogni numero che compare nella lista deve corrispondere all'inizio di una subroutine, cioè avete bisogno di un'istruzione RETURN per tornare alla linea che segue immediatamente ON GOSUB.
Se il valore di *espressione* è 0 o maggiore del numero di elementi della lista (ma minore o uguale a 255), il BASIC prosegue dalla prima istruzione eseguibile.

## **ON KEY (n)** Istruzione

Definisce il numero della linea a cui il BASIC trova la routine di intercettamento nel caso in cui venga premuto il tasto funzione indicato o un tasto di controllo cursore. È utilizzabile solo in Advanced BASIC o Compiled BASIC.

*Sintassi:*
ON KEY(*esprnum*) GOSUB *linea*

*Esempio:*

```
ON KEY(4) GOSUB 1000
```

Il parametro *esprnum* deve essere compreso tra 1 e 20 ed indica quale tasto dev'essere intercettato, secondo questo schema:

| | |
|---|---|
| 1-10 | Tasti funzione da F1 a F10 |
| 11 | Cursore in alto (↑) |
| 12 | Cursore a sinistra (←) |
| 13 | Cursore a destra (→) |
| 14 | Cursore in basso (↓) |
| 15-20 | Tasti definiti nella forma: KEY *esprnum*, CHR$ (*stato-shift*)+CHR$ (*codice di scansione*) |

Il parametro *linea* è il numero della linea a cui inizia la routine di intercettamento per il tasto indicato; porre *linea* uguale a 0 disabilita l'intercettamento di quel tasto.
Per attivare quest'istruzione deve prima essere eseguita un'istruzione

KEY(*esprnum*) ON. Se, dopo l'istruzione KEY(*esprnum*) ON, viene specificato in ON KEY(*esprnum*) un numero di linea diverso da zero, ogni volta che il programma esegue una nuova istruzione il BASIC controlla se il tasto specificato era stato premuto: se è così, il BASIC esegue un'istruzione GOSUB alla linea specificata in *linea*.
Se invece viene eseguita un'istruzione KEY(*esprnum*) OFF, non viene predisposto nessun intercettamento per il tasto indicato, ed inoltre, se il tasto viene premuto, l'evento non viene memorizzato.
Anche nel caso in cui venga eseguita un'istruzione KEY(*esprnum*) STOP non vengono predisposti intercettamenti per il tasto indicato; comunque, se il tasto viene premuto, l'evento viene memorizzato, cossicché non appena viene eseguita un'istruzione KEY(*esprnum*) ON, ha occorrenza un intercettamento.
Quando si verifica un intercettamento, automaticamente viene eseguita un'istruzione KEY(*esprnum*) STOP, cosicché non possono mai avvenire intercettamenti recursivi. L'esecuzione dell'istruzione RETURN dalla routine di intercettamento provoca a sua volta l'esecuzione automatica dell'istruzione KEY(*esprnum*) ON, a meno che non fosse già stata eseguita un'istruzione KEY(*esprnum*) OFF all'interno della routine stessa.
Non possono avvenire eventi di intercettamento se il BASIC non sta eseguendo un programma. Quando viene intercettato un errore (risultato dell'istruzione ON ERROR), tutti gli intercettamenti vengono automaticamente disabilitati, inclusi ERROR, STRIG(*n*), PEN, COM(*n*), PLAY(*n*), TIMER e KEY(*n*).
L'intercettamento dei tasti non può operare quando vengono premuti altri tasti prima di quello specificato. Il tasto che ha causato l'intercettamento non può essere esaminato con le istruzioni INPUT$ e INKEY$, cosicché la routine di intercettamento per ogni tasto dev'essere differente se la funzione desiderata è differente.
Potete usare l'istruzione RETURN *linea* per tornare al programma BASIC ad un numero di linea fissato. Comunque questo ritorno non locale deve essere usato con cura, poiché qualunque altro GOSUB, WHILE, FOR che sia attivo al momento dell'intercettamento, rimarrà attivo comunque.
L'istruzione KEY(*esprnum*) ON non ha effetto su quei valori dei tasti di funzione definibili visualizzati nella parte bassa dello schermo.

## **ON PEN** Istruzione

Assegna un numero di linea a cui il BASIC trasferisce il controllo quando viene attivata la penna ottica. Valida solo in Advanced BASIC ed in Compiled BASIC.

*Sintassi:*
ON PEN GOSUB *linea*

*Esempio:*

```
ON PEN GOSUB 6000
```

Il parametro *linea* è il numero di linea di inizio della routine di intercettamento: assegnandole zero si disabilita l'intercettamento della penna ottica.
Per attivare quest'istruzione deve prima essere eseguita un'istruzione PEN ON. Se, dopo l'istruzione PEN ON, viene specificato in ON PEN un numero di linea diverso da zero, ogni volta che il programma esegue una nuova istruzione, il BASIC controlla se la penna ottica è stata attivata: se è così, il BASIC esegue un'istruzione GOSUB alla linea specificata in *linea*.
Se invece viene eseguita un'istruzione PEN OFF, non viene predisposto nessun intercettamento, ed anche se la penna ottica viene attivata, l'evento non viene memorizzato.
Non vengono predisposti intercettamenti anche nel caso in cui venga eseguita un'istruzione PEN STOP; comunque l'attività della penna viene memorizzata, in modo che ha occorrenza un intercettamento non appena viene eseguita un'istruzione PEN ON.
Quando si verifica un intercettamento, automaticamente viene eseguita un'istruzione PEN STOP. L'esecuzione dell'istruzione RETURN dalla routine di intercettamento provoca a sua volta l'esecuzione automatica dell'istruzione PEN ON, a meno che non fosse già stata eseguita un'istruzione PEN OFF all'interno della routine stessa.
Non possono avvenire eventi di intercettamento se il BASIC non sta eseguendo un programma. Quando viene intercettato un errore (risultato dell'istruzione ON ERROR), tutti gli intercettamenti vengono automaticamente disabilitati, inclusi ERROR, STRIG(*n*), PEN, COM(*n*), PLAY(*n*), TIMER e KEY(*n*).
Non è assegnato il valore PEN(0) quando l'attività della penna causa un intercettamento. Potete usare l'istruzione RETURN *linea* per tornare al programma BASIC ad un numero di linea fissato. Comunque, questo ritorno non locale deve essere usato con cura, poiché un altro GOSUB, WHILE, FOR che sia attivo al momento dell'intercettamento, rimarrà attivo comunque.

## **ON PLAY(n)** Istruzione

Abilita l'intercettamento di errori a seconda dello stato del buffer di accompagnamento musicale; permette che un accompagnamento con-

tinuo segua l'esecuzione di un programma. Utilizzabile solo in Advanced BASIC.

*Sintassi:*
ON PLAY(*n*) GOSUB *linea*

*Esempio:*

```
ON PLAY(4) GOSUB 1000
```

Il parametro *n* è un'espressione intera variabile tra 1 e 32, che indica il numero di note che devono rimanere nel buffer per dare inizio alla routine di intercettamento. Valori al di fuori di questo intervallo causeranno un errore ILLEGAL FUNCTION CALL. Il parametro *linea* è il numero di linea di inizio della routine di intercettamento: assegnandole zero si disabilita l'intercettamento della routine suono.
Per attivare quest'istruzione deve prima essere eseguita un'istruzione PLAY ON. Se, dopo l'istruzione PLAY ON, viene specificato in PLAY(*n*) un numero di linea diverso da zero, ogni volta che il programma esegue una nuova istruzione il BASIC controlla se il buffer di accompagnamento è passato da *n* a $n-1$ note: se è così, il BASIC esegue un'istruzione GOSUB alla linea specificata in *linea*.
Se invece viene eseguita un'istruzione PLAY OFF, non viene predisposto nessun intercettamento, ed anche qualora si verificasse, l'evento non verrebbe memorizzato. Anche nel caso in cui venga eseguita un'istruzione PLAY STOP, non vengono predisposti intercettamenti; comunque, l'attività musicale viene memorizzata, cosicché scatta un intercettamento non appena viene eseguita un'istruzione PLAY ON.
Quando si verifica un intercettamento, automaticamente viene eseguita un'istruzione PLAY STOP, per impedire intercettamenti recursivi. L'esecuzione dell'istruzione RETURN dalla routine di intercettamento provoca a sua volta l'esecuzione automatica dell'istruzione PLAY ON, a meno che non fosse già stata eseguita un'istruzione PLAY OFF all'interno della routine stessa.
Non possono avvenire eventi di intercettamento se il BASIC non sta eseguendo un programma. Quando viene intercettato un errore (risultato dell'istruzione ON ERROR), tutti gli intercettamenti vengono automaticamente disabilitati, inclusi ERROR, STRIG(*n*), PEN, COM(*n*), PLAY(*n*), TIMER e KEY(*n*).
Potete usare l'istruzione RETURN *linea* per tornare al BASIC ad un numero di linea fissato. Comunque questo ritorno non locale deve essere usato con cura, poiché qualunque altro GOSUB, WHILE, FOR che sia attivo al momento dell'intercettamento, rimarrà attivo comunque.

NOTE:

1. Una routine di intercettamento può essere eseguita solo quando il modo dell'istruzione PLAY è MB (accompagnamento). Non viene invece eseguita quando il modo è MF.
2. Non viene eseguita la routine di intercettamento se il buffer di accompagnamento è vuoto al momento dell'esecuzione dell'istruzione PLAY ON.
3. Se il parametro *n* è molto grande, si verificheranno così tanti eventi da lasciare poco tempo per eseguire il resto del programma: fate perciò attenzione nella scelta del valore *n*.

Per maggiori dettagli vedere la funzione PLAY(*n*).

## **ON STRIG (n)** Istruzione

Definisce il numero della linea a cui il BASIC trova la routine di intercettamento nel caso in cui venga premuto il pulsante di un joystick. È utilizzabile solo in Advanced BASIC o Compiled BASIC.

*Sintassi:*
ON STRIG (*pulsante*) GOSUB *linea*

*Esempio:*

```
ON STRIG(4) GOSUB 1000
```

Il parametro *pulsante* può valere 0, 2, 4 oppure 6 ed indica quale tasto dev'essere intercettato, secondo questo schema:

| | |
|---|---|
| 0 | pulsante A1 |
| 2 | pulsante B1 |
| 4 | pulsante A2 |
| 6 | pulsante B2 |

Il parametro *linea* è il numero della linea a cui inizia la routine di intercettamento per il pulsante indicato; porre *linea* uguale a 0 disabilita l'intercettamento di quel pulsante.
Per attivare quest'istruzione deve prima essere eseguita un'istruzione STRIG(*pulsante*) ON. Se, dopo l'istruzione STRIG(*pulsante*) ON, viene specificato in ON STRIG(*pulsante*) un numero di linea diverso da zero, ogni volta che il programma esegue una nuova istruzione il BASIC controlla se il pulsante specificato era stato premuto: se è così, il BASIC esegue un'istruzione GOSUB alla linea specificata in *linea*.

Se invece viene eseguita un'istruzione STRIG(*pulsante*) OFF, non viene predisposto nessun intercettamento per il pulsante indicato, ed inoltre, anche se il pulsante viene premuto, l'evento non viene memorizzato.
Anche nel caso in cui venga eseguita un'istruzione STRIG(*pulsante*) STOP, non vengono predisposti intercettamenti per il pulsante indicato; comunque, se il pulsante viene premuto, l'evento viene memorizzato, cossicché non appena viene eseguita un'istruzione STRIG(*pulsante*) ON, ha occorrenza un intercettamento.
Intercettamenti recursivi non possono mai avvenire perché quando si verifica un intercettamento, automaticamente viene eseguita un'istruzione STRIG(*pulsante*) STOP.
L'esecuzione dell'istruzione RETURN dalla routine di intercettamento provoca a sua volta l'esecuzione automatica dell'istruzione STRIG(*pulsante*) ON, a meno che non fosse già stata eseguita un'istruzione STRIG(*pulsante*) OFF all'interno della routine stessa.
Non possono avvenire eventi di intercettamento se il BASIC non sta eseguendo un programma. Quando viene intercettato un errore (risultato dell'istruzione ON ERROR), tutti gli intercettamenti vengono automaticamente disabilitati, inclusi ERROR, STRIG(*n*), PEN, COM(*n*), PLAY(*n*), TIMER e KEY(*n*).
L'uso di STRIG(*pulsante*) ON attiverà la routine di intercettamento per controllare lo stato del pulsante del joystick indicato. Eventuali pressioni del pulsante che causano l'avvio di routine di intercettamento non definiscono però il valore delle funzioni STRIG(0), STRIG(2), STRIG(4), STRIG(6). Potete usare l'istruzione RETURN *linea* per tornare al BASIC ad un numero di linea fissato. Comunque questo ritorno non locale deve essere usato con cura, poiché qualunque altro GOSUB, WHILE, FOR che sia attivo al momento dell'intercettamento, rimarrà attivo comunque.

## **ON TIMER** Istruzione

Trasferisce il controllo ad un dato numero di linea di un programma BASIC dopo un determinato intervallo di tempo. L'istruzione ON TIMER è utile per produrre intervalli di tempo in un programma. È utilizzabile solo in Advanced BASIC.

*Sintassi:*

ON TIMER (*esprnum*) GOSUB *linea*

*Esempio:*

```
ON TIMER(60) GOSUB 1000
```

Il parametro *esprnum* indica il numero di secondi che devono trascorrere perché sia intercettato il successivo evento ON TIMER. L'intervallo di variabilità per *esprnum* è da 1 a 86400 (da 1 secondo a 24 ore); valori all'esterno di quest'intervallo producono l'errore ILLEGAL FUNCTION CALL. Il parametro *linea* è il numero della linea a cui inizia la routine di intercettamento; porre *linea* uguale a 0 disabilita l'intercettamento del contatore. Osservate che l'istruzione ON TIMER non modifica il valore della funzione TIMER, che restituisce ancora il numero di secondi trascorsi dalla mezzanotte o dall'ultimo avviamento del sistema.
Per attivare quest'istruzione deve prima essere eseguita un'istruzione TIMER ON. Se, dopo l'istruzione TIMER ON, viene specificato in ON TIMER(*esprnum*) un numero di linea diverso da zero, ogni volta che il programma esegue una nuova istruzione il BASIC tiene il conteggio dei secondi trascorsi. Quando è trascorso il numero di secondi indicato, il BASIC esegue un'istruzione GOSUB alla linea specificata in *linea*. Dopo un evento il BASIC continua a contare il numero di secondi, fino al successivo intercettamento.
Se invece viene eseguita un'istruzione TIMER OFF, non viene predisposto nessun intercettamento, ed inoltre, anche se è trascorso il numero di secondi indicato, l'evento non viene memorizzato. Anche nel caso in cui venga eseguita un'istruzione TIMER STOP, non vengono predisposti intercettamenti; comunque viene memorizzato ogni evento, cosicché non appena viene eseguita un'istruzione TIMER ON, ha occorrenza un intercettamento.
Quando si verifica un intercettamento, automaticamente viene eseguita un'istruzione TIMER STOP, per evitare intercettamenti recursivi. L'esecuzione dell'istruzione RETURN dalla routine di intercettamento provoca a sua volta l'esecuzione automatica dell'istruzione TIMER ON, a meno che non fosse già stata eseguita un'istruzione TIMER OFF all'interno della routine stessa.
Non possono avvenire eventi di intercettamento se il BASIC non sta eseguendo un programma. Quando viene intercettato un errore (risultato dell'istruzione ON ERROR), tutti gli intercettamenti vengono automaticamente disabilitati, inclusi ERROR, STRIG(*n*), PEN, COM(*n*), PLAY(*n*), TIMER e KEY(*n*).
Potete usare l'istruzione RETURN *linea* per tornare al programma BASIC ad un numero di linea fissato. Comunque questo ritorno non locale deve essere usato con cura, poiché qualunque GOSUB, WHILE, FOR che sia attivo al momento dell'intercettamento, rimarrà attivo comunque.

## **OPEN** (ALT O) Istruzione

Consente operazioni di I/O verso file o dispositivi esterni.

*Sintassi:*

OPEN "*specfile*" [FOR *modo*] AS [#] *numfile* [LEN=*lungrec*]

oppure

OPEN *modo2*, [#] *numfile*, "*specifile*" [,*lungrec*]

*Esempi:*

```
OPEN "A:FILEDATI" FOR OUTPUT AS #1
OPEN "PI" AS #2 LEN=256
OPEN MODE$, #1, "B:TEMP", 256
OPEN "R", #4, "\LVL1\LVL2\DATI"
```

Il parametro *modo*, che appare nella prima forma può essere:

| | |
|---|---|
| OUTPUT | specifica il modo di output sequenziale |
| INPUT | specifica il modo di input sequenziale |
| APPEND | specifica il modo di output sequenziale dove il puntatore del file viene posto alla fine dei dati quando il file viene aperto. |

Si noti che *modo* deve essere una costante stringa non racchiusa tra virgolette, e se viene omesso si assume per default l'accesso diretto.
Il parametro *modo2*, nella seconda forma, è un'espressione stringa il cui primo carattere deve essere uno dei seguenti:

| | |
|---|---|
| O | specifica il modo di output sequenziale |
| I | specifica il modo di input sequenziale |
| R | specifica il modo di input/output ad accesso diretto |

Per quanto riguarda invece i seguenti parametri, presenti in entrambe le forme:

*numfile* è un'espressione intera appartenente all'intervallo compreso tra 1 ed il numero massimo di file permessi, che di default è 3, ma che può essere cambiato con l'opzione /F: nel comando BASIC.

*lungrec* è un'espressione numerica che, se presente, assegna la lunghezza del record per il file ad accesso diretto. Può avere valori compresi nell'intervallo da 1 a 32767, con default di 128 byte per record; comunque non può mai superare il valore dell'opzione /S: nel comando BASIC.

L'istruzione OPEN definisce un buffer per le operazioni di I/O del file o del dispositivo esterno e determina il modo di accesso che sarà usato per il buffer.
Il parametro *numfile* è il numero che viene associato al file finché questo

rimane aperto, ed è usato da altre istruzioni di I/O per fare riferimento al file o al dispositivo. Un'istruzione OPEN deve essere eseguita prima di ogni operazione di I/O verso il file o il dispositivo fatta per mezzo di istruzioni o funzioni che richiedono il numero del file o ognuna delle seguenti istruzioni:

| | | | |
|---|---|---|---|
| PRINT# | WRITE# | INPUT# | GET |
| PRINT#USING | INPUT$ | LINE INPUT# | PUT |

Le istruzioni GET e PUT sono utilizzabili per file ad accesso diretto (o file di comunicazione — vedi OPEN "COM...). Un file su disco può essere sia sequenziale che ad accesso diretto, ed una stampante può essere aperta sia nel modo sequenziale che nel modo ad accesso diretto; comunque tutti gli altri dispositivi possono essere aperti solo per operazioni sequenziali.
Il BASIC solitamente aggiunge un carattere LF (*line feed*) dopo ogni ritorno a capo (CHR$(13)) inviato alla stampante. Comunque se aprite un file per la stampante (LPT1:, LPT2:, LPT3:) come file ad accesso diretto con dimensione 255 byte, questo carattere LF viene soppresso.
APPEND è utilizzabile solo per file su disco. Il puntatore al file inizialmente è posto alla fine del file ed il numero di record è posto uguale all'ultimo record del file. Le istruzioni PRINT# e WRITE#, in questo caso, aggiungeranno i dati alla fine del file.

> NOTA: ogni tanto è possibile avere un particolare file aperto con più di un numero identificativo. Ciò permette di usarlo in modi diversi per differenti scopi; oppure, per chiarezza di programma, potete usare differenti numeri di file per differenti modi di accesso. Ad ogni numero di file è associato un buffer diverso, e voi dovrete porre attenzione se scrivete usando un numero di file e leggete usandone un altro. Un file che sia già stato aperto non può essere aperto per output sequenziali né può essere aggiunto ad altri file.

Se viene omesso il nome del dispositivo, viene assunto il drive di default del DOS. Non potete aprire più di tre file contemporaneamente, a meno che non usiate l'opzione /F nel comando BASIC per riservare spazio di memoria per più file.
Se un file aperto per un input in realtà non esiste, si incorre nell'errore FILE NOT FOUND. Se invece il file inesistente viene aperto per un output, per essere aggiunto, o per un accesso diretto, viene creato un file.
Ogni valore assegnato al di fuori dell'intervallo indicato causerà l'errore ILLEGAL FUNCTION CALL ed il file non verrà aperto.
Si noti infine che è possibile aprire un file sequenziale e assegnare la lunghezza del record.

## OPEN "COM... Istruzione

Apre un file per comunicazioni. Non è utilizzabile senza l'interfaccia per comunicazioni asincrone ACA.

*Sintassi:*

OPEN "COM*n*:[*velocità*] [,*parità*] [,*bit dati*] [,*bit stop*]
[,RS] [,CS[*n*]] [,DS[*n*]] [,CD[*n*]] [,LF] [,PE]"
AS [#]*numfile* [LEN=*numero*]

*Esempi:*

```
OPEN "COM1:" AS 1
OPEN "COM1:2400" AS #2
OPEN "COM2:9600,N,8,,CS,DS,CD" AS #1
OPEN "COM1:1200,N,8,,CS10000,DS10000,CD10000,LF" AS #1
```

Il parametro *n*, che può valere 1 o 2, indica il numero dell'ACA; *velocità* è una costante intera che specifica il baud rate di ricezione/trasmissione in bit per secondo (b/s): i valori consentiti per *velocità* sono 75, 110, 150, 300, 600, 1200, 1800, 2400, 4800 e 9600 ed il valore di default è 300 b/s; *parità* è una costante di un carattere che specifica la parità per la trasmissione e la ricezione nel modo seguente:

- S SPACE: il bit di parità viene sempre trasmesso e ricevuto come uno spazio (0)
- O ODD: parità di trasmissione dispari, controllo della parità in ricezione dispari
- M MARK: il bit di parità viene sempre trasmesso e ricevuto come un bit di segnale (1)
- E EVEN: parità di trasmissione pari, controllo della parità in ricezione pari
- N NONE: nessuna parità in trasmissione, nessun controllo di parità in ricezione.

Il valore di default è EVEN (E).
Il parametro *bit dati* è una costante intera che indica il numero di bit di dati da trasmettere o ricevere, da scegliere tra 3, 4, 5, 6, 7 oppure 8, con default 7; *bit stop* è una costante intera indicante il numero di bit di stop, 1 o 2: il valore di default è 2 bit di stop per 75 e 110 b/s, 1 per tutti gli altri. Se usate 4 o 5 per *bit dati*, un 2 qui significherà 1 bit e 1/2; *numfile* è un intero che ha come valore un numero di file permesso. Il numero viene allora associato al file per tutto il tempo che il file resta aperto e viene usato da altre istruzioni di I/O per riferirsi al file; *numero* infine è il

massimo numero di byte che possono essere letti dal buffer di comunicazione usando GET e PUT. Il valore di default è 128 byte.
L'istruzione OPEN "COM... assegna un buffer per l'I/O esattamente come OPEN fa per i file su dischetto. È in grado di supportare comunicazioni con altri computer e periferiche per mezzo della linea RS232 asincrona. Un dispositivo di comunicazione può essere aperto solo con un numero di file per volta.
Le opzioni RS, CS, DS, CD, LF, e PE agiscono sui segnali di linea nei modi seguenti:

| | |
|---|---|
| RS | sopprime RTS (Request To Send) |
| CS[*n*] | controlla CTS (Clear to Send) |
| DS[*n*] | controlla DSR (Data Set Ready) |
| CD[*n*] | controlla CD (Carrier Detect) |
| LF | invia un line feed dopo ogni CR |
| PE | abilita il controllo di parità |

Il CD è anche noto come RLSD (Received Line Signal Detect).

NOTA: i parametri *velocità, parità, bit dati* e *bit stop* sono opzionali, mentre non lo sono RS, CS, DS, CD, LF, e PE.

La linea RTS viene accesa quando voi eseguite un'istruzione OPEN "COM... a meno che non includiate l'opzione RS.
L'argomento *n* in CS, DS e CD specifica il numero di millisecondi di attesa del segnale prima della restituzione del messaggio d'errore DEVICE TIMEOUT; *n* deve appartenere all'intervallo da 0 a 65535 e se viene omesso o posto uguale a zero, non viene controllato per intero lo stato della linea.
I valori di default sono CS 1000, DS 1000 e CD 0; se invece viene indicato RS, il valore di default per CS è 0. Istruzioni di I/O indirizzate ad un file di comunicazione normalmente non hanno effetto se i segnali CTS o DSR non sono presenti; il sistema attende per un secondo prima di dare il messaggio DEVICE TIMEOUT. Le opzioni CS e DS permettono di trascurare queste linee o di specificare l'intervallo di tempo che deve trascorrere prima del segnale di *timeout*.
Usualmente il Carrier Detect (CD o RLSD) viene ignorato nell'esecuzione di un'istruzione OPEN "COM... L'opzione CD permette di controllare questa linea, nello stesso modo di CS e DS, aggiungendo il parametro *n*; se *n* è tralasciato o uguale a zero, il Carrier Detect non viene controllato (equivale a tralasciare l'opzione CD).
Il parametro LF viene utilizzato, in questi file di comunicazione, come mezzo per stampare su stampanti seriali; se indicate LF, viene automaticamente aggiunto un carattere line feed (LF; Hex 0A) dopo ogni ritorno

carrello (CR; Hex 0C), compresi quelli determinati dalla definizione della larghezza. Osservate che, quando vengono utilizzate per leggere file di comunicazione aperti con l'opzione LF, le istruzioni INPUT# e LINE INPUT# si fermano non appena incontrano un ritorno carrello: in questo modo il carattere LF viene ignorato.
L'opzione PE abilita il controllo della parità che per default è disabilitato. L'opzione PE provoca un errore di I/O ogni volta che trova errori di parità e commuta il valore del bit di ordine più alto quando il numero di bit è 7 o meno. L'opzione PE non influenza errori di *framing* o *overrun* che, a loro volta pongono sempre a 1 il valore del bit di ordine più alto e causano un errore di I/O.
Ogni errore di battitura all'interno della stringa di espressioni che inizia con *velocità* ha come effetto un errore BAD FILE NAME, ma non viene indicato quale parametro sia errato.
Se specificate che il numero di bit di dati dev'essere 8, dovete indicare come parità N, cioè nessuna parità, mentre se ne specificate 4, la parità dev'essere definita, cioè non è valido il parametro N. Il BASIC utilizza tutti e otto i bit di un byte per memorizzare i numeri e perciò se trasmettete o ricevete dati numerici (ad esempio, per mezzo dell'istruzione PUT) dovete specificare 8 come numero di bit di dati. (Non è valida l'affermazione nel caso in cui inviate i dati numerici sotto forma di testo). Vedi l'istruzione OPEN per l'apertura di dispositivi diversi da quelli di comunicazione.

## **OPTION BASE** Istruzione

Definisce il valore minimo per gli indici degli array.

*Sintassi:*
OPTION BASE *esprnum*

*Esempio:*

```
OPTION BASE 1
```

Il parametro *esprnum* vale 1 o 0; il valore di default è 0. Se viene eseguita l'istruzione

```
OPTION BASE 1
```

il più basso valore che un indice di array può assumere è 1. L'istruzione OPTION BASE dev'essere utilizzata sempre prima di definire o usare un qualsiasi array.

## **OUT** Istruzione

Invia un byte a una porta di output.

*Sintassi:*
OUT *porta, byte*

*Esempio:*

```
OUT 32,100
```

Il parametro *porta* è un'espressione numerica che indica il numero della porta di output e deve appartenere all'intervallo da 0 a 65535; *byte* invece è l'espressione numerica per il dato da trasmettere e deve appartenere all'intervallo da 0 a 255. Per la descrizione dei numeri di porta utilizzabili (indirizzi di I/O) fate riferimento al manuale IBM *Technical Reference.*
L'istruzione OUT è complementare alla funzione INP (vedi INP).
Uno degli usi di OUT è la gestione del video. Su alcuni monitor collegati con la scheda Colore/Grafica, potreste notare che due o tre caratteri della linea non vengono visualizzati sullo schermo. Se il vostro video non dispone del controllo per la centratura, per spostare il video potete usare l'istruzione:

```
OUT 980,2:OUT 981,43
```

che permette di ottenere uno scorrimento a destra del video di due caratteri, per lo schermo con larghezza 40 colonne (o 16 punti nel modo Grafico in media risoluzione, o 32 punti nel modo Grafico in alta risoluzione) o l'istruzione:

```
OUT 980,2:OUT 981,85
```

che farà scorrere il video di 5 caratteri a destra nello schermo con larghezza di 80 colonne.
Lo scorrimento prodotto da queste istruzioni OUT rimarrà valido fino all'esecuzione di un'istruzione WIDTH o SCREEN. Per far scorrere il video come descritto può anche essere usato il comando MODE del DOS; quest'ultimo ha il vantaggio di rimanere valido fino all'esecuzione di una reinizializzazione del sistema.

## **PAINT** Istruzione

Riempie un'area dello schermo col colore selezionato. Utilizzabile in Compiled BASIC ed in Advanced BASIC, solo nel modo Grafico.

*Sintassi:*

PAINT (*x,y*) [[,*colore*] [,*contorno*] [,*fondo*]]

*Esempio:*

```
PAINT (50,50),1,2
```

I parametri (*x,y*) sono le coordinate di un punto all'interno dell'area che deve essere riempita, che possono essere date sia in forma assoluta che in forma relativa; il punto di coordinate (*x,y*) sarà usato come punto di partenza. Il parametro *colore* è il colore usato per riempire l'area di schermo; in media risoluzione questo colore è selezionato dalla tavolozza corrente come definita nell'istruzione COLOR; al numero 0 corrisponde il colore del fondo. Il valore di default per il colore del testo è 3. Invece in alta risoluzione, *colore* uguale a 0 indica nero, uguale a 1 indica bianco, che è anche il colore di default. Se *colore* è un'espressione di tipo stringa, allora viene compiuta un'operazione di riempimento, come viene descritto più avanti. Il parametro *contorno* è il colore del bordo della figura da riempire, con valore nell'intervallo prima descritto; il valore di default è *colore*. Infine *fondo* è un'espressione di tipo stringa usata nel colore di riempimento.

Poiché in alta risoluzione vi sono solo due colori disponibili, non ha senso che i parametri *colore* e *contorno* siano differenti; *contorno* è posto per default uguale a *colore* e non è necessario il terzo parametro. In media risoluzione possiamo riempire la figura con il colore 1 e colorare il contorno con il colore 2; un esempio potrebbe essere un cerchio verde con il bordo rosso.

Il punto di partenza di PAINT deve essere all'interno della figura da disegnare. Il bordo della figura a sua volta deve avere lo stesso colore di contorno. Se il punto di coordinate (*x,y*) è già colorato con lo stesso colore del bordo, PAINT non avrà alcun effetto. Se viene omesso il parametro *colore* viene usato il colore del testo (3 in media risoluzione, 1 in alta risoluzione). L'istruzione PAINT può dipingere ogni tipo di figura, ma eventuali contorni frastagliati della figura aumenteranno lo spazio di stack richiesto da PAINT. Così, se devono essere eseguite delle operazioni di pittura molto complesse, potete usare l'istruzione CLEAR all'inizio del programma per aumentare lo spazio di stack disponibile.

L'uso del colore di riempimento richiede che il parametro *colore* sia

un'espressione di tipo stringa che definisca una maschera di riempimento; quest'espressione stringa ha la forma:

CHR$(&H*hex*)+CHR$(&H*hex*)+...

dove ogni espressione CHR$ definisce una riga della maschera; la stringa ne può contenere al massimo 64. La struttura della maschera è:

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | x aumenta → bit del byte del motivo | | | | | |
| x,y | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | |
| 0,0 | x | x | x | x | x | x | x | x | Byte 0 del motivo |
| 0,1 | x | x | x | x | x | x | x | x | Byte 1 del motivo |
| : | | | | | | | | | |
| 0,63 | x | x | x | x | x | x | x | x | Byte 63 del motivo |

Il motivo viene ripetuto uniformemente sopra l'intero schermo. Ogni byte della stringa maschera 8 bit lungo l'asse X quando disegna dei punti; inoltre viene ruotato a richiesta per allineare l'asse Y, così:

motivo_byte_maschera = y mod motivo_lunghezza

In modo Grafico ad alta risoluzione ogni bit della maschera corrisponde ad un pixel dello schermo. Un punto viene disegnato per ogni posizione della maschera che ha valore 1. Poiché ogni byte della maschera specifica una riga del motivo ed ogni bit corrisponde ad un pixel dello schermo, ogni byte è abilitato a disegnare 8 punti attraverso lo schermo.
In modo Grafico a media risoluzione, due bit della maschera corrispondono ad un singolo pixel dello schermo. Ogni coppia di bit descrive uno dei 4 possibili colori associati ad ognuno dei pixel da disegnare. Così in media risoluzione, ogni byte del motivo descrive 4 pixel. Nella tavolozza di colori 0, i colori verde, rosso, marrone corrispondono rispettivamente ai numeri binari 01, 10, 11; nella tavolozza 1, i colori cyan, magenta e bianco corrispondono rispettivamente ai numeri binari 01, 10, 11.
Nel modo Grafico sia in alta risoluzione che in media risoluzione, il numero 0 non provoca alcun cambiamento nel pixel associato; in qualunque stato si trovi precedentemente il pixel (on/off in alta risoluzione, uno dei quattro colori in media risoluzione), il processo del colore di riempimento non lo cambierà.
Potreste voler colorare un'area già dipinta con un motivo a righe dello stesso colore. Normalmente incontrare punti dello stesso colore del riempimento in atto, costituisce una condizione di termine dell'esecuzione. Potete usare il parametro *fondo* per superare questa condizione. Non po-

tete comunque specificare più di due righe nel motivo uguali al parametro *fondo*, poiché così facendo si incorrerebbe nell'errore ILLEGAL FUNCTION CALL.

## **PEEK** Funzione

Restituisce il byte letto nella locazione di memoria specificata.

*Sintassi:*
*byte* = PEEK(*n*)

Il parametro *n* è un intero appartenente all'intervallo da 0 a 65535, ed è il valore di offset dal segmento corrente come definito dall'istruzione DEF SEG che indica l'indirizzo della locazione di memoria da leggere. Il valore restituito, *byte*, sarà un intero tra 0 e 255.
L'istruzione PEEK è complementare all'istruzione POKE.

## **PEN** Funzione

Legge la posizione della penna ottica.

*Sintassi:*
*varnum* = PEN(*esprnum*)

*Esempio:*

```
D=PEN(0)
```

Il parametro *esprnum*, che deve appartenere all'intervallo da 0 a 9, può dare i seguenti risultati:

0 Un flag che indica che la penna è stata attivata dopo l'ultima richiesta. Restituisce −1 se era spenta, 0 se non lo era.
1 Restituisce il valore della coordinata x del punto dove la penna era stata attivata l'ultima volta. L'intervallo è da 0 a 319 in media risoluzione, da 0 a 639 in alta risoluzione.
2 Restituisce il valore della coordinata y del punto dove la penna era stata attivata l'ultima volta. L'intervallo è da 0 a 199.
3 Restituisce il valore corrente della penna, −1 se è premuto, 0 in caso contrario.
4 Restituisce l'ultima coordinata valida conosciuta x. L'intervallo è da 0 a 319 in media risoluzione, da 0 a 639 in alta risoluzione.

5 Restituisce l'ultima coordinata valida conosciuta y. L'intervallo è da 0 a 199.
6 Restituisce il carattere posizione di riga dove la penna era stata attivata l'ultima volta. L'intervallo è da 0 a 24.
7 Restituisce il carattere posizione di colonna dove la penna era stata attivata l'ultima volta. L'intervallo è da 1 a 40, o da 1 a 80, secondo il valore di WIDTH.
8 Restituisce l'ultimo carattere posizione di riga valido conosciuto. L'intervallo è da 1 a 24.
9 Restituisce l'ultimo carattere posizione di colonna valido conosciuto. L'intervallo è da 1 a 40, o da 1 a 80, secondo il valore di WIDTH.

## **PEN** Istruzione

Abilita e disabilita la penna ottica.

*Sintassi:*
PEN ON
PEN OFF
PEN STOP (solo in Advanced e Compiled BASIC)

L'istruzione PEN ON abilita la funzione PEN, che inizialmente è OFF. L'istruzione PEN ON deve essere eseguita prima di una chiamata della funzione di lettura con la penna, altrimenti verrà visualizzato il messaggio d'errore ILLEGAL FUNCTION CALL.
D'altra parte, per migliorare la velocità di esecuzione, è buona norma spegnere la penna con l'istruzione PEN OFF quando non la state usando.
In Advanced BASIC, eseguendo l'istruzione PEN ON si potrà predisporre l'intercettamento con l'istruzione ON PEN.
Dopo l'esecuzione di PEN ON, se nell'istruzione OPEN era stato specificato un numero di linea diverso da zero, ogni volta che il programma esegue una nuova istruzione, il BASIC controlla se la penna è stata attivata.
L'istruzione PEN OFF disabilita la funzione di lettura della penna. In Advanced BASIC non viene predisposto l'intercettamento della penna; quando predisposto, non viene ricordata l'azione della penna ottica.
L'istruzione PEN STOP infine, utilizzabile solo in Advanced BASIC, disabilita l'intercettamento dell'attività della penna ottica, ma viene ricordata l'azione eventuale in modo che, quando viene eseguita l'istruzione PEN ON, ha immediatamente luogo un intercettamento. Quando la penna si trova ai bordi dello schermo, il valore restituito non è accurato.

## **PLAY** Istruzione

Suona un brano musicale come specificato da *stringa*. Utilizzabile solo in Advanced BASIC ed in Compiled BASIC.

*Sintassi:*
PLAY "*stringa*"

*Esempio:*

```
PLAY "MB T100 O3 L8 XMARY$;P8 FFF4 GB-B-4 XMARY$;
     GFFGFE-."
dove MARY$="GFE-FGGG"
```

PLAY implementa un concetto simile a quello dell'istruzione DRAW: inserire un *linguaggio di definizione della musica* in una stringa di caratteri interpretabili come programma musicale. Il parametro *stringa* è un'espressione stringa che consiste di più comandi musicali a carattere singolo.
Questi comandi sono:

Lettere dalla A alla G con gli opzionali #, +, −. Suonano le note indicate (in notazione anglosassone) nell'ottava corrente. Un # e + indicano un diesis ed un segno − indica un bemolle. I caratteri #, +, −, non sono consentiti a meno che non corrispondano a dei tasti neri del pianoforte. Per esempio, B# non è una nota consentita.

O *n* Assegna l'ottava corrente per le note che seguono. Le ottave sono 7, numerate da 0 a 6, ed ogni ottava va dalla nota C (Do) alla nota B (Si). L'ottava 4 è l'ottava di default.

> *n* Passa all'ottava successiva verso l'alto e suona la nota. Le note seguenti saranno suonate nella nuova ottava finché non verrà richiesto un nuovo cambio. Se l'ottava corrente è l'ottava 6, non vi saranno cambiamenti. Il comando è di fatto un cambio di ottava relativo: quale sia la nuova ottava dipende da quale era la vecchia. Per esempio, PLAY ">G" alza l'ottava e suona la nota Sol.

< *n* Passa all'ottava successiva verso il basso e suona la nota. Le note seguenti saranno suonate nella nuova ottava finché non verrà richiesto un nuovo cambio. Se l'ottava corrente è l'ottava 0, non vi saranno cambiamenti. Come il precedente, è un cambio di ottava relativo.

N *n* Suona la nota *n*, che può essere un valore compreso tra 0 e 84. Vi sono 84 possibili note in 7 ottave diverse; *n* uguale a 0 corrisponde

ad una pausa. È un modo alternativo per selezionare le note, oltre a quello di specificare l'ottava (O *n*) ed il nome della nota (da A a G).

L *n* Assegna la durata alle note che seguono. La durata effettiva della nota è 1/*n*, con un *n* da 0 a 64. La seguente tabella chiarisce la funzione L *n*:

| **Comando** | **Durata equivalente** |
|---|---|
| L1 | Nota intera |
| L2 | Mezza nota |
| L3 | Una di una tripletta di tre mezze note (1/3 della battuta di 4/4) |
| L4 | 1/4 di nota |
| L5 | Una di una quintupletta (1/5 di battuta) |
| L6 | Una di una tripletta di 1/4 di nota |
| : | |
| L64 | 1/64 di nota |

L'indicazione può anche seguire la nota se si vuole cambiare la durata solo per quella nota. Per esempio A16 è equivalente a L16A.

P *n* Pausa. Il parametro *n*, nell'intervallo da 1 a 64, rappresenta la durata della pausa allo stesso modo di L.

Il punto quando segue una nota, la fa suonare per una durata uguale a 3/2 quella indicata. Dopo una nota può esserci più di un punto, e la sua lunghezza verrà calcolata di conseguenza. Per esempio, "A.." sarà suonata con una lunghezza uguale ai 9/4 della lunghezza specificata in L, "A..." con una lunghezza di 27/8, e così via. I punti possono apparire anche dopo P (pausa) per aumentare allo stesso modo la durata della pausa.

T *n* Tempo. Assegna il numero di note da 1/4 da suonare in un minuto, con *n* che va da 32 a 255; il valore di default è 120. Vedere SOUND per la tabella dei tempi comuni e gli equivalenti battiti per minuto.

MF Musica solista (Music Foreground). È la musica (creata da SOUND e PLAY) suonata come solista. Cioè ogni nota o suono successivo non viene suonato finché non è terminata la precedente nota. La musica solista è quella di default.

MB Musica di accompagnamento (Music Background). È la musica (creata da SOUND e PLAY) suonata come accompagnamento al posto della musica solista. Ogni nota o suono viene inserito in un buffer, consentendo al programma BASIC di continuare l'esecuzione di programmi mentre suona una musica di accompagnamento. Come accompagnamento possono essere suonate al massimo 32 note, o pause, per volta.

MN Musica normale. Ogni nota viene suonata per i 7/8 della lunghezza specificata da L. MN è l'assegnamento di default fra i tre possibili (MN, ML, MS).

ML Legato. Ogni nota viene suonata per tutta la lunghezza specificata da L.

MS Staccato. Ogni nota viene suonata per i 3/4 della lunghezza specificata da L.

X *variabile*; Esegue stringhe specificate.

In tutti questi comandi l'argomento *n* può essere una costante o=*variabile*;, dove *variabile* è il nome di una variabile. Il carattere ";" è necessario quando usate una variabile e quando usate il comando X. Il ";" è altrimenti opzionale tra comandi, eccetto dopo MF, MB, MN, ML, o MS dove non è permesso. Ogni spazio bianco in *stringa* viene ignorato.
Potete anche specificare le variabili nella forma VARPTR$(*variabile*) invece che nelle forma =*variabile*;. Ciò è utile in programmi che in seguito verranno compilati. Per esempio:

| **Metodo A** | **Metodo B** |
|---|---|
| PLAY "XA$;" | PLAY "X"+VARPTR$ (A$) |
| PLAY "O=I;" | PLAY "O="+VARPTR$(I) |

Potete usare X per memorizzare dei ritornelli in una stringa e chiamarli in modo ripetitivo, con tempi ed ottave differenti, dall'interno di altre stringhe.

## **PLAY (n)** Funzione

Restituisce il numero corrente delle note nel buffer della musica di accompagnamento. Utilizzabile solo in Advanced BASIC.

*Sintassi:*
*n* = PLAY(*esprnum*)

*Esempio:*

```
IF PLAY(O)=3 GOTO 1000
```

Il parametro *esprnum* è un argomento fittizio e può avere ogni valore. Il massimo valore che può essere restituito è 32, il massimo numero di note che può essere contenuto nel buffer. Il conteggio delle note viene restituito solo nel caso che si stia usando il modo Accompagnamento (MB) (vedi

PLAY). Se il programma viene avviato nel modo musica solista (MF), la funzione PLAY restituisce il valore 0.

## **PMAP** Funzione

Trasforma e ritrasforma tra loro coordinate fisiche ed esterne. Utilizzabile solo in Advanced BASIC.

*Sintassi:*
*varnum* = PMAP(*coord,esprnum*)

*Esempio:*

```
FISICHE_X=PMAP(ESTERNE_X,0)
```

La funzione PMAP trasforma le coordinate *x* e *y* da coordinate fisiche di sistema (come definite dall'istruzione WINDOW) in esterne, e viceversa. Il parametro *coord* rappresenta le coordinate del punto che deve essere disegnato; *esprnum* deve appartenere all'intervallo da 0 a 3, cosicché:

*esprnum*=0 riporta la coordinata esterna *x* alla coordinata fisica *x*
*esprnum*=1 riporta la coordinata esterna *y* alla coordinata fisica *y*
*esprnum*=2 riporta la coordinata fisica *x* alla coordinata esterna *x*
*esprnum*=3 riporta la coordinata fisica *y* alla coordinata esterna *y*

## **POINT** Funzione

Restituisce il colore di uno specificato punto dello schermo. Utilizzabile solo in modo Grafico.

*Sintassi:*
*varnum*=POINT(*x,y*)
*varnum*=POINT (*esprnum*)

*Esempio:*

```
IF POINT (R,C)=3 THEN 100
```

(*x,y*) sono le coordinate del punto da usare; devono essere in forma assoluta.
Se il punto dato è al di fuori dell'intervallo consentito, viene restituito il valore −1. In media risoluzione valori validi restituiti sono 0, 1, 2, e 3, mentre in alta risoluzione sono 0 e 1.

Se il parametro dell'istruzione POINT è un singolo valore (*esprnum*), verrà restituito il valore della coordinata grafica corrente *x* o *y*, che è il solito LRP del modo Grafico. Il parametro *esprnum* può avere un valore compreso tra 0 e 3, con i seguenti significati:

0 restituisce la coordinata fisica corrente *x*
1 restituisce la coordinata fisica corrente *y*
2 se è attiva l'istruzione WINDOW, POINT restituisce la coordinata corrente esterna *x*, altrimenti la coordinata fisica corrente *y* (come per *esprnum*=0 se WINDOW non è attiva)
3 se è attiva l'istruzione WINDOW, POINT restituisce la coordinata corrente esterna *y*, altrimenti la coordinata fisica corrente *y* (come per *esprnum*=0 se WINDOW non è attiva).

Vedere WINDOW per ulteriori informazioni.

## **POKE** Istruzione

Scrive un byte in una locazione di memoria.

*Sintassi:*
POKE *n, byte*

*Esempio:*

```
POKE 106,0
```

Il parametro *n*, che deve appartenere all'intervallo da 0 a 65535, indica l'indirizzo della locazione di memoria dove il dato deve essere scritto; è un offset del segmento corrente come definito dall'istruzione DEF SEG. Il parametro *byte*, invece, è il dato da scrivere nella locazione specificata e deve appartenere all'intervallo da 0 a 255.
La funzione complementare di POKE è PEEK. Le istruzioni PEEK e POKE sono utili per un'efficiente memorizzazione di dati, caricamento di subroutine in linguaggio macchina e trasferimento di argomenti e risultati da e verso subroutine in linguaggio macchina.
Attenzione: il BASIC non compie alcun controllo sull'indirizzo; perciò non eseguite POKE nello stack del BASIC, nell'area riservata per le variabili, o nei vostri programmi.

## **POS** Funzione

Restituisce la posizione di colonna corrente del cursore.

*Sintassi:*
*varnum*=POS(*esprnum*)

*Esempio:*

```
IF POS(0)>40 THEN PRINT CHR$(13)
```

Il parametro *esprnum* è un argomento fittizio. Viene restituita la posizione orizzontale corrente (colonna) del cursore, che deve essere un valore all'interno dell'intervallo da 1 a 40, o da 1 a 80, secondo il valore di WIDTH. Può essere usata l'istruzione CSRLIN per trovare la posizione verticale (riga) del cursore.

## **PRINT** (ALT P) Istruzione

Visualizza dati sullo schermo.

*Sintassi:*
PRINT [*lista di espressioni*] [;] [,]
?[*lista di espressioni*] [;] [,]

*Esempi:*

```
PRINT "Chi sei ?";
PRINT
?35*2*.8
PRINT "X=" X;"Y=" Y
PRINT A,B,C
```

Il parametro *lista di espressioni* è una lista di espressioni numeriche o a stringa o entrambe, separate da virgole, spazi bianchi, o punto e virgola. Ogni costante di tipo stringa della lista deve essere racchiusa tra virgolette. Se la *lista di espressioni* viene omessa, viene visualizzata una linea bianca. Se viene inclusa una lista di espressioni numeriche, sullo schermo verrà visualizzato il valore dell'espressione.
La posizione di ogni termine scritto è determinata dalla punteggiatura usata per i termini nella lista. Il BASIC divide la linea in zone di stampa di 14 caratteri ciascuna. Nella *lista di espressioni* una virgola provoca la visualizzazione del valore successivo all'inizio della zona successiva; un punto e virgola invece provoca la visualizzazione del valore immediatamente dopo il precedente. Scrivere le espressioni separandole con spazi bianchi ha lo stesso effetto che scrivere un punto e virgola.
Se al termine della *lista di espressioni* si trova una virgola, un punto e virgola, un SPC o un TAB, l'istruzione PRINT successiva inizia a stampa-

re sulla stessa linea, rispettando le spaziature. Se invece la lista di espressioni termina senza nessuno di questi caratteri, alla fine della linea viene eseguito un CR, cioè il BASIC muove il cursore all'inizio della linea seguente.
Se la lunghezza del valore da stampare supera il numero di caratteri rimanenti sulla linea corrente, il valore sarà visualizzato all'inizio della linea successiva. Se il valore da visualizzare è più lungo del valore definito per WIDTH, il BASIC scrive quanto più possibile sulla linea corrente e continua a scrivere il resto del valore sulla linea fisica successiva.
I numeri visualizzati sono sempre seguiti da uno spazio; i numeri positivi sono sempre preceduti da uno spazio, mentre i numeri negativi sono sempre preceduti dal segno meno −. I numeri in precisione semplice possono essere rappresentati con non più di 7 cifre in formato a virgola fissa con la stessa accuratezza di quanto potrebbero essere rappresentati in formato a virgola mobile se fossero inviati in output con la virgola fissa o il formato intero. Per esempio, 10^(−7) è inviato in output come 0.0000001, e 10^(−8) è inviato in output come 1E−8.
Il BASIC inserisce automaticamente una sequenza di CR/LF dopo aver visualizzato 40 o 80 caratteri come definito dall'istruzione WIDTH. Ciò provocherà il salto di 2 linee quando visualizzerete esattamente 40 (o 80) caratteri a meno che l'istruzione PRINT non termini con un punto e virgola. L'istruzione LPRINT viene utilizzata per inviare informazioni alla stampante (vedi LPRINT e LPRINT USING).

## **PRINT USING** (ALT P ALT U) Istruzione

Stampa stringhe o numeri usando uno speciale formato.

*Sintassi:*
PRINT USING "*stringa*"; *lista di espressioni* [;] [,]

*Esempi:*

```
PRINT USING "##.##";M%
PRINT USING "$$###.##";D
```

Il parametro *stringa* consiste di speciali caratteri di formattazione (descritti di seguito), che determinano il campo e il formato delle stringhe e dei numeri da stampare. Il parametro *lista di espressioni* consiste di espressioni numeriche o di tipo stringa che devono essere stampate, separate da virgole o da punti e virgola.

## Campi alfanumerici

Quando si usa PRINT USING per stampare stringhe, deve essere utilizzato uno di questi tre caratteri per formattare il campo delle stringhe:

! Indica che solo il primo carattere della stringa deve essere stampato.

\ **n** \ spazi Indica che n+2 caratteri della stringa devono essere stampati. Se i due backslash non includono spazi, vengono stampati due caratteri; con uno spazio ne vengono stampati tre e così via. Se la stringa è più lunga del campo, i caratteri in eccesso vengono ignorati. Se il campo è più lungo della stringa, questa viene giustificata a sinistra nel campo e vengono inseriti spazi a destra, come mostrato nell'esempio che segue.

```
10 A$="ECCO": B$="QUI"
20 PRINT USING "!";A$;B$
30 PRINT USING "\  \";A$;B$
40 PRINT USING "\    \";A$;B$;"!!"
RUN
EQ
ECCOQUI
ECCO  QUI   !!
```

& Indica un campo per stringhe di lunghezza variabile. Se il campo viene specificato con il simbolo &, infatti, la stringa è inviata in output esattamente come è stata ricevuta in input.

```
10 A$="ECCO": B$="QUI"
20 PRINT USING "!";A$;
30 PRINT USING "&";B$
RUN
EQUI
```

## Campi numerici

Quando PRINT USING è usata per scrivere numeri, possono essere utilizzati i seguenti caratteri speciali per definire il campo numerico:

\# Rappresenta la posizione di ogni cifra; queste posizioni vengono sempre riempite. Se il numero ha meno cifre di quelle specificate per il campo di stampa, viene giustificato a destra (e preceduto da spazi) nel campo.

Un punto decimale può essere inserito in una qualsiasi posizione del campo. Se la stringa di formato indica che almeno una cifra deve precedere il punto, questa verrà sempre scritta (anche 0 se necessario). I numeri vengono opportunamente arrotondati.

```
PRINT USING "##.##";.78
 0.78
```

```
PRINT USING "###.##";987.564
987.56
```

```
PRINT USING "##.##";  10.2,5.3,66.789,.234
10.20 5.3066.79 0.23
```

Nell'ultimo esempio sono stati inseriti tre spazi alla fine della stringa di formato perché i valori risultassero separati sulla linea di output.

\+ Un segno + all'inizio o alla fine della stringa di formato fa sì che il segno (positivo o negativo) venga scritto prima o dopo il numero.

– Un segno – alla fine del formato fa sì che in coda ai numeri negativi venga scritto un segno meno.

```
PRINT USING "+##.##  ";-68.95,2.4,55.6,-.9
-68.95  +2.40 +55.60  -0.90
```

```
PRINT USING"##.##-";-68.95,-7.01,22.449
68.95- 7.01-22.45
```

** Un doppio asterisco all'inizio della stringa di formato fa sì che gli spazi che precedono il numero siano riempiti di asterischi; inoltre indica la posizione per altre due eventuali cifre.

```
PRINT USING "**#.# ";12.39,-0.9,765.1
*12.4 *-0.9 765.1
```

$$ Un doppio segno di dollaro fa sì che il simbolo $ venga scritto appena prima del numero formattato. Indica due ulteriori posizioni

per il numero, una delle quali è il simbolo $. Non può essere utilizzata in questo caso la forma esponenziale, ed i numeri negativi possono essere presenti solo se il segno meno segue il numero, anziché precederlo.

```
PRINT USING "$$###.##";456.78
 $456.78
```

**$ La sequenza **$ all'inizio della stringa di formato combina gli effetti dei due precedenti simboli. Gli spazi che precedono il numero vengono riempiti con asterischi ed il simbolo $ viene scritto appena prima del numero. Indica inoltre tre ulteriori posizioni per le cifre del numero, una delle quali è occupata dal simbolo $.

```
PRINT USING "**$##.##";2.34
***$2.34
```

Una virgola alla sinistra del punto decimale nella stringa di formato indica che una virgola dev'essere scritta alla sinistra di ogni terza cifra alla sinistra del punto decimale. Una virgola alla fine della stringa di formato, invece, viene scritta come parte della stringa. La virgola riserva un'ulteriore posizione per le cifre del numero. Una virgola all'inizio della stringa di formato scrive semplicemente una virgola all'inizio della stringa, mentre non ha effetto con il formato esponenziale.

```
PRINT USING "####,.##";1234.5
1,234.50
```

```
PRINT USING "####.##,";1234.5
1234.50,
```

```
PRINT USING ",####.##,";1234.5
,1234.50,
```

^^^^ Quattro accenti circonflessi posti dopo i caratteri che rappresentano le posizioni delle cifre indicano il formato esponenziale e riservano lo spazio per scrivere una delle due notazioni E±nn e

D±nn. Per il punto decimale può essere indicata una qualsiasi posizione; le cifre significative vengono giustificate a sinistra e l'esponente calcolato di conseguenza. Tranne nel caso in cui venga indicato un + all'inizio della stringa di formato, oppure un più o un meno alla fine, viene lasciata una posizione alla sinistra del punto decimale per uno spazio o per il segno meno.

```
PRINT USING "##.##^^^^";234.56
 2.35E+02
```

```
PRINT USING ".###^^^^-";-88888
.889E+05-
```

```
PRINT USING "+.##^^^^";123
+.12E+03
```

_ Una sottolineatura nella stringa di formato fa sì che il carattere successivo sia scritto come è presente nella stringa.

```
PRINT USING "_!##.##_!";12.34
!12.34!
```

La sottolineatura stessa può divenire un carattere da visualizzare, se viene posto in questa sequenza: "__" nella stringa di formato.

Se il numero da scrivere è più grande di quello che può essere contenuto nel campo numerico indicato, viene aggiunto un segno % (per cento) davanti al numero scritto. Anche se il numero eccede il campo a causa dell'arrotondamento viene aggiunto il segno % davanti al numero arrotondato.

```
PRINT USING "##.##";111.22
%111.22
```

```
PRINT USING ".##";.999
%1.00
```

Se il numero di cifre indicate supera 24, si verifica un errore ILLEGAL FUNCTION CALL.

## PRINT# e PRINT#USING Istruzioni

Scrivono sequenzialmente i dati in un file.

*Sintassi:*
PRINT #*numfile*, [USING "*stringa*";] *lista di espressioni*

*Esempi:*

```
PRINT #1,A,B,C$
PRINT #1, USING B$;A,C,Z$
```

Il parametro *numfile* è lo stesso numero utilizzato per l'apertura del file per l'output. *stringa* è composta dai caratteri di formato come descritto nell'istruzione PRINT USING. *lista di espressioni* è un elenco delle espressioni numeriche o di tipo stringa, o di entrambe, che devono essere scritte nel file.
L'istruzione PRINT# non comprime i dati nel file. Un'immagine dei dati viene riportata nel file proprio come sarebbe apparsa sullo schermo in seguito ad un'istruzione PRINT; per questa ragione, abbiate cura a delimitare con esattezza i dati all'interno del file in modo da poterli poi leggere correttamente.
Nella *lista di espressioni* le espressioni numeriche possono essere separate da punto e virgola; per esempio:

```
PRINT #1,A;B;C;X;Y;Z
```

Se vengono usate le virgole come separatori, gli spazi bianchi in sovrappiù inseriti tra i campi di stampa vengono scritti anche nel file.
Le espressioni di tipo stringa devono essere separate da punto e virgola nell'elenco. Per definire correttamente le stringhe nel file, utilizzate delimitatori espliciti nella *lista di espressioni*. Supponiamo ad esempio che A$="CAMERA" e B$="93604-1". L'istruzione

```
PRINT #1,A$;B$
```

scriverà allora nel file CAMERA93604-1; poiché non ci sono delimitatori, questa non può più essere letta come due stringhe separate. Per correggere, inserite un delimitatore esplicito nell'istruzione PRINT #:

```
PRINT #1,A$;",";B$
```

L'immagine scritta nel file diviene quindi:

```
CAMERA,93604-1
```

che ora può essere letta nelle due stringhe separate.
Se la stringa stessa contiene virgole, punto e virgola, spazi bianchi significativi, ritorni carrello, a capo, scriveteli nel file racchiusi esplicitamente tra virgolette ottenute per mezzo di CHR$(34). Se abbiamo A$="OROLOGIO, AUTOMATICO" e B$=68789−1", l'istruzione

```
PRINT #1,A$;B$
```

scrive nel file la seguente immagine

```
OROLOGIO, AUTOMATICO 68789-1
```

e l'istruzione

```
INPUT #1,A$,B$
```

leggerà la stringa "OROLOGIO" come A$ e "AUTOMATICO 68789−1" come B$.
Per separare queste stringhe in modo corretto all'interno del file, scrivete le virgolette utilizzando CHR$(34). L'istruzione:

```
PRINT #1,CHR$(34);A$;CHR$(34);CHR$(34);B$;CHR$(34)
```

scrive nel file:

```
"OROLOGIO, AUTOMATICO"" 68789-1"
```

e l'istruzione:

```
INPUT #1,A$,B$
```

legge "OROLOGIO, AUTOMATICO" come A$ e "68789−1" come B$.
L'istruzione PRINT# può anche essere usata con l'opzione USING per controllare il formato del file, come mostrato in questo esempio:

```
PRINT #1,USING "$$###.##,";J;K;L
```

Il modo più semplice per evitare problemi è utilizzare l'istruzione WRITE# al posto dell'istruzione PRINT#.

## PSET e PRESET Istruzioni

Disegnano un punto nella posizione dello schermo indicata. Sono eseguibili solo in modo Grafico.

*Sintassi:*
PSET (*x,y*) [,*colore*]
PRESET (*x,y*) [,*colore*]

*Esempi:*

```
PSET (50,4),2
PRESET (100,100)
```

(*x,y*) rappresentano le coordinate del punto da disegnare; possono essere sia in forma relativa che in forma assoluta. *colore* indica il colore da usare, nell'intervallo da 0 a 3. In media risoluzione *colore* sceglie il colore dalla tavolozza corrente definita dall'istruzione COLOR; 0 rappresenta il colore del fondo; il colore del testo è il numero 3. In alta risoluzione il colore 0 indica il nero, mentre 1 indica il bianco, colore di default; il valore 2 viene considerato come 0 e 3 come 1.
Se per *colore* non viene indicato alcun parametro, PRESET usa per default il colore di fondo, mentre PSET usa il colore del testo; in questo modo, senza indicazione di *colore*, PSET disegna i punti e PRESET li cancella. Se invece viene indicato il parametro *colore*,non c'è differenza tra le due istruzioni.
Se viene fornita alle istruzioni PSET e PRESET una coordinata al di fuori dello schermo, non viene effettuata alcuna azione e non si verifica neppure un errore. Un valore di *colore* maggiore di 3, invece, produrrà un errore ILLEGAL FUNCTION CALL.

## PUT (File) Istruzione

Scrive in un file ad accesso casuale o in un buffer di comunicazioni un record letto da un buffer ad accesso casuale.

*Sintassi:*
PUT [#]*numfile* [,*numero record*]

*Esempi:*

```
PUT #1
PUT #2,15
```

Il parametro *numfile* è il numero con cui il file era stato aperto; *numero record* rappresenta il numero del record che deve essere scritto, nell'intervallo tra 1 e 16 777 215. Se viene omesso, al record viene assegnato il successivo numero di record disponibile (dopo l'ultima istruzione PUT). Per inserire dei caratteri nel buffer di un file ad accesso diretto prima di un'istruzione PUT, possono essere usate PRINT#, PRINT# USING, WRITE#, LSET e RSET. Nel caso di WRITE#, il BASIC riempie il buffer con spazi fino al carattere CR.
Ogni tentativo di leggere oltre la fine del file provocherà la visualizzazione del messaggio d'errore FIELD OVERFLOW. Poiché il BASIC ed il DOS racchiudono il massimo numero possibile di record in settori di 512 byte, l'istruzione PUT non necessariamente realizza una scrittura fisica sul dischetto.
L'istruzione PUT può essere usata per file di comunicazione; in questo caso il parametro *numero record* rappresenta il numero di byte da scrivere nel file di comunicazione. Questo numero deve essere minore od uguale al valore dall'opzione LEN nell'istruzione OPEN "COM...

## **PUT** (Grafica) Istruzione

Disegna dei colori in una specificata area dello schermo. Utilizzabile solo in Advanced BASIC ed in Compiled BASIC, in modo Grafico.

*Sintassi:*
PUT (*x,y*),*vettoreimmagine*[,*operatore*]

*Esempi:*

```
PUT (12,12), PIC
PUT (50,20), D, PRESET
PUT (1,1), Z, AND
```

Il parametro (*x,y*) rappresenta le coordinate dell'angolo in basso a sinistra dell'immagine da trasferire; *vettoreimmagine* è il nome di un array numerico contenente le informazioni da trasferire (vedere GET (Grafica) per ulteriori informazioni su questo array); *operatore* invece può essere uno dei seguenti:

PSET
PRESET
XOR

OR
AND

dove XOR è il valore di default.
L'istruzione PUT è l'opposto di GET nel senso che prende dati dall'array e li porta sullo schermo; comunque, esegue l'opzione di interazione, per mezzo dell'azione con il dato già sullo schermo.
PSET come azione, semplicemente memorizza il dato dall'array allo schermo; si può quindi definire il vero opposto di GET.
PRESET è uguale a PSET, tranne nel fatto che produce un'immagine negativa; cioè un valore 0 nell'array provoca la colorazione del punto col colore numero 3, e viceversa: un valore 1 nell'array produce una colorazione con il colore 2, e viceversa.
AND viene usato per trasferire l'immagine solo se sotto l'immagine trasferita esiste già un'altra immagine.
OR viene usato per sovrapporre l'immagine all'immagine già esistente.
XOR è un modo speciale che può essere usato per l'animazione. Infatti provoca l'inversione dei punti dello schermo dove esiste un punto nel vettore immagine. XOR ha una proprietà unica che lo rende utile per l'animazione: quando si esegue due volte un'istruzione PUT con un'immagine su un fondo complesso, il fondo viene ripristinato immutato. Questo vi permette di far muovere un oggetto senza cancellare il fondo.
In media risoluzione AND, XOR e OR hanno i seguenti effetti sui colori:

**AND**

**Valore dell'array**

| | | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|
| | 0 | 0 | 0 | 0 | 0 |
| **Schermo** | 1 | 0 | 1 | 0 | 1 |
| | 2 | 0 | 0 | 2 | 2 |
| | 3 | 0 | 1 | 2 | 3 |

**OR**

**Valore dell'array**

| | | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|
| | 0 | 0 | 1 | 2 | 3 |
| **Schermo** | 1 | 1 | 1 | 3 | 3 |
| | 2 | 2 | 3 | 2 | 3 |
| | 3 | 3 | 3 | 3 | 3 |

**XOR**

**Valore dell'array**

| | | 0 | 1 | 2 | 3 |
|---|---|---|---|---|---|
| | 0 | 0 | 1 | 2 | 3 |
| **Schermo** | 1 | 1 | 0 | 3 | 2 |
| | 2 | 2 | 3 | 0 | 1 |
| | 3 | 3 | 2 | 1 | 0 |

L'animazione di un oggetto può essere realizzata nel modo seguente:

1. Eseguire un'istruzione PUT di un oggetto sullo schermo con l'opzione XOR
2. Ricalcolare la nuova posizione dell'oggetto sullo schermo
3. Eseguire un'istruzione PUT dell'oggetto sullo schermo con l'opzione XOR una seconda volta nella vecchia locazione per rimuovere la vecchia immagine
4. Tornare al punto 1, questa volta eseguendo l'istruzione PUT dell'oggetto nella nuova posizione

I movimenti compiuti in questo modo lasciano lo schermo inalterato. Può essere ridotto il tremolio minimizzando il tempo tra il punto 4 ed il punto 1 e facendo attenzione a che vi sia un certo tempo di ritardo tra il punto 1 ed il punto 3. Se si vuole animare più di un oggetto, ognuno di essi deve essere trattato come descritto sopra, passo dopo passo.
Se non è importante mantenere inalterato il fondo, l'animazione può essere realizzata con l'opzione PSET. Comunque, dovrete ricordare di avere un'area di immagine che possa contenere l'immagine "prima" e "dopo" dell'oggetto, poiché l'area supplementare cancellerà effettivamente la vecchia immagine. Questo metodo può essere un poco più veloce di quello realizzato con l'istruzione XOR prima descritto, poiché per muovere l'oggetto viene richiesta una sola istruzione PUT (anche se voi dovete tracciare un'immagine più grande).
Se l'immagine da trasferire è troppo grande per stare nello schermo, si incorrerà nell'errore ILLEGAL FUNCTION CALL.

## **RANDOMIZE** Istruzione

Definisce un nuovo seme da cui generare dei numeri casuali.

*Sintassi:*

RANDOMIZE [*esprint*]
RANDOMIZE TIMER

*Esempio:*

```
RANDOMIZE (LEN(NA$))
```

L'intero *esprint* sarà usato come seme del numero casuale; se viene omesso, il BASIC sospende l'esecuzione del programma e richiede un valore visualizzando

```
RANDOMIZE
Random number seed (-32768 to 32767)?
```

prima di eseguire RANDOMIZE.
Se il generatore di numeri casuali non viene ridefinito, la funzione RND restituisce la stessa sequenza di numeri casuali ogni volta che il programma viene avviato. Per cambiare questa sequenza, basta porre un'istruzione RANDOMIZE all'inizio del programma e cambiare il seme ad ogni avvio.
Usando l'istruzione RANDOMIZE TIMER, potete ottenere un seme differente senza che vi venga richiesto ogni volta di inserirlo. La funzione TIMER restituisce un numero in precisione semplice che rappresenta il numero di secondi dalla mezzanotte o dall'ultima reinizializzazione del sistema. Questa funzione restituisce un valore differente ogni volta che viene eseguita un'istruzione RANDOMIZE TIMER, creando così un nuovo valore del seme.

## **READ** Istruzione

Legge valori dalle istruzioni DATA e li assegna alle variabili (vedi DATA).

*Sintassi:*
READ *variabile* [,*variabile*]...

*Esempio:*

```
READ A$,B
```

Il parametro *variabile* è una variabile numerica o a stringa o un array di elementi che riceve i valori letti dalle tabelle DATA.
Un'istruzione READ deve essere sempre usata in unione con un'istruzione DATA. L'istruzione READ assegna i valori dell'istruzione DATA alle variabili contenute nell'istruzione READ in rapporto uno a uno. Le variabili dell'istruzione READ, che possono essere numeriche o a stringa, de-

vono avere assegnati dei valori dello stesso loro tipo, altrimenti si incorre nell'errore SYNTAX ERROR.
Una singola istruzione READ può accedere ad una o più istruzioni DATA (rispettandone l'ordine), o più istruzioni READ possono accedere alla stessa istruzione DATA. Se il numero di variabili nella lista presente nell'istruzione READ eccede il numero di elementi nell'istruzione DATA, le successive istruzioni READ cominceranno a leggere dati a partire dall'ultimo elemento non letto. Se invece non vi sono altre istruzioni READ, vengono ignorati i dati in eccesso. Per rileggere dati da ogni riga della lista dell'istruzione DATA, bisogna usare l'istruzione RESTORE (vedi RESTORE).

## **REM** Istruzione

Inserisce in un programma dei commenti esplicativi.

*Sintassi:*
REM *commento*
*'commento*

*Esempio:*

```
REM calcola la velocità media
GOSUB 1000 ' traccia le coordinate
```

Il parametro *commento* può essere una sequenza di caratteri. L'istruzione REM non viene eseguita, ma stampata, esattamente come era stata scritta, quando viene listato il programma; rallenta minimamente il tempo di esecuzione ed occupa spazio di memoria. Le istruzioni REM possono essere raggiunte usando un'istruzione GOSUB o GOTO: l'esecuzione continuerà con la prima istruzione eseguibile dopo l'istruzione REM.
I commenti possono essere aggiunti alla fine di una linea preceduti da un apice invece che da REM. Se inserite un commento in una linea con altre istruzioni, il commento deve essere l'ultima istruzione della linea.

## **RENUM** Comando

Rinumera le linee di programma. Non è utilizzabile in Compiled BASIC.

*Sintassi:*
RENUM [*nuovonum*] [,[*vecchionum*] [,*incr*]]

*Esempi:*

```
RENUM
RENUM,,15
RENUM 100
RENUM 100,50,5
```

Il parametro *nuovonum* è il primo numero di linea che deve essere usato nella sequenza, con un valore di default di 10; *vecchionum* è la linea del programma corrente da cui deve partire la rinumerazione, ed il valore di default è la prima linea del programma. Il parametro *incr* è l'incremento da usare nella nuova sequenza, con un valore di default di 10.
L'istruzione RENUM inoltre cambia tutti i numeri di linea contenuti nelle istruzioni GOTO, GOSUB, THEN, ELSE, ON-GOTO, ON-GOSUB, RESTORE, RESUME e ERL, affinché riflettano i cambiamenti avvenuti. Se dopo una di queste istruzioni appare una linea inesistente, verrà visualizzato il messaggio d'errore UNDEFINED LINE NUMBER xxxxx IN yyyyy. Il numero di linea non corretto cui ci si riferisce (xxxxx) non può essere cambiato da RENUM, mentre può essere cambiato il numero di linea yyyyy.

NOTA: l'istruzione RENUM non può essere usata per cambiare l'ordine delle linee di programma (per esempio, RENUM 15,30 quando il programma ha tre linee numerate 10, 20, e 30), o per creare numeri di linea maggiori di 65529, altrimenti si incorre nell'errore ILLEGAL FUNCTION CALL.

## **RESET** Comando

Chiude tutti i file su disco e cancella il buffer del sistema.

*Sintassi:*
RESET

*Esempio:*

```
RESET
```

Se tutti i file aperti sono su disco, RESET ha lo stesso effetto di CLOSE senza il numero di file.

## RESTORE Istruzione

Consente di rileggere le istruzioni DATA a partire da una linea specificata.

*Sintassi:*
RESTORE [*linea*]

*Esempi:*

```
RESTORE
RESTORE 100
```

Il parametro *linea* è numero di linea dell'istruzione DATA nel programma. Dopo l'esecuzione dell'istruzione RESTORE, la successiva istruzione READ accede al primo termine della prima istruzione DATA nel programma. Se invece viene specificato il parametro *linea*, la successiva istruzione READ accede al primo termine dell'istruzione DATA che si trova alla linea specificata.

## RESUME Istruzione

Continua l'esecuzione di un programma dopo che è stata eseguita una procedura di recupero di errore.

*Sintassi:*
RESUME [0]
RESUME NEXT
RESUME *linea*

*Esempi:*

```
RESUME
RESUME NEXT
RESUME 100
```

Può essere usata ciascuna delle sintassi precedentemente mostrate, secondo il punto dove viene ripresa l'esecuzione.

RESUME o RESUME 0

L'esecuzione riprende all'istruzione che ha causato l'errore.

NOTA: se voi provate a rinumerare un programma contenente un'istruzione RESUME 0, causerete l'errore UNDEFINED LINE ERROR. L'istruzione continuerà ad essere RESUME 0, che è accettabile.

RESUME NEXT

L'esecuzione riprende all'istruzione immediatamente seguente quella che ha causato l'errore.

RESUME *linea*

L'esecuzione riprende alla *linea* specificata.
Un'istruzione RESUME che non è una routine di intercettamento di un errore, causa il messaggio d'errore RESUME WITHOUT ERROR.

## **RETURN** Istruzione

Conclude una subroutine e invia il controllo all'istruzione successiva a quella che aveva chiamato la subroutine.

*Sintassi:*
RETURN [*linea*]

*Esempio:*

```
RETURN
```

Il parametro *linea* è il numero della linea di programma alla quale volete tornare. Potete usarla solo in Advanced BASIC ed in Compiled BASIC. L'aggiunta del parametro *linea* consente un ritorno indirizzato da una routine di intercettamento di un evento. Da una di queste routine potrete tornare sovente al numero di linea fissato eliminando nello stesso tempo il GOSUB creato dall'intercettamento. Il ritorno non locale deve comunque essere usato con attenzione, poiché ogni altra GOSUB, WHILE, o FOR che sia attiva al momento dell'intercettamento, rimarrà attiva.

## RIGHT$ Funzione

Restituisce il numero di caratteri specificati più a destra di *stringa.*

*Sintassi:*
*varstr*=RIGHT$(*stringa,n*)

*Esempio:*

```
PRINT RIGHT$(A$,7)
```

Il parametro *n* specifica il numero di caratteri che devono essere nel risultato e, se è maggiore od uguale a LEN(*stringa*), viene restituita l'intera stringa. Se *stringa* è zero, viene restituita la stringa nulla.

## RMDIR Comando

Rimuove un directory dal disco specificato. Utilizzabile solo in Disk BASIC ed in Advanced BASIC.

*Sintassi:*
RMDIR *"nomedir"*

*Esempio:*

```
RMDIR "VENDITE\AGOSTO"
```

Il parametro *nomedir* è un'espressione di tipo stringa che identifica il directory da rimuovere dalla struttura esistente, e non deve superare una lunghezza di 63 caratteri (vedere il Capitolo 3 per ulteriori informazioni sui cataloghi con struttura ad albero).
Il directory specificato in *nomedir* deve essere svuotato di tutti i file e subdirectory prima di essere rimosso, con l'eccezione di "." e di "..", o si incorrerà nel PATH/FILE ACCESS ERROR. (Vedere i comandi MKDIR e CHDIR per la creazione ed il cambiamento di directory).

## RND Funzione

Restituisce un numero casuale tra 0 ed 1.

*Sintassi:*
*varnum*=RND[(*esprnum*)]

*Esempio:*

```
X=INT(RND*(6+1))
```

Il parametro *esprnum* è un'espressione numerica che influisce sul valore restituito nel modo descritto più avanti. La stessa sequenza di numeri casuali viene generata ogni volta che il programma viene avviato senza che il generatore di numeri casuali venga ridefinito. Questo è molto facile da fare con l'istruzione RANDOMIZE. Potete anche definire il nuovo seme del generatore quando chiamate la funzione RND usando il parametro *esprnum*, con *esprnum* negativa. Questo genera spesso una particolare sequenza per quella *esprnum*; questa sequenza non viene influenzata da RANDOMIZE, cosicché se volete generare una differente sequenza ogni volta che il programma viene avviato, dovete usare ogni volta un differente valore per *esprnum*.
Se *esprnum* è positiva o non indicata, RND(*esprnum*) genera il successivo numero casuale della sequenza. RND(0) ripete l'ultimo numero generato. Per ottenere numeri casuali nell'intervallo tra 0 ed *n*, usate la formula

INT((*n*+1)*RND)

## **RUN** (ALT R) Comando

Avvia l'esecuzione di un programma.

*Sintassi:*
RUN [*linea*]
RUN "*specfile*"[,R]

*Esempi:*

```
RUN
RUN 50
RUN "TEST",R
```

Il parametro *linea* è il numero di linea del programma in memoria da cui volete iniziare l'esecuzione. Per i file su disco viene utilizzata per default l'estensione .BAS.
RUN o RUN *linea* avviano l'esecuzione del programma corrente in memoria. Se *linea* viene specificata, l'esecuzione inizia alla linea indicata, altrimenti inizia alla linea con il numero più basso.
RUN *specfile* carica in memoria un file da disco e lo avvia; chiude tutti i file aperti e cancella il contenuto corrente della memoria prima di cari-

care il programma indicato. Comunque, con l'opzione R restano aperti tutti i file di dati.
Eseguendo un comando RUN si elimina ogni suono in esecuzione e si reinizializza la musica solista. Infine, alle istruzioni PEN e STRIG viene assegnato il valore OFF.

## **SAVE** Comando

Salva un file di programma BASIC su disco. Non utilizzabile in Compiled BASIC.

*Sintassi:*

SAVE "*specfile*" [,A] [,P]

*Esempi:*

```
SAVE "A:DATI",A
SAVE "MESE"
```

L'estensione .BAS viene aggiunta al nome del file se questo è lungo 8 o meno caratteri, e non viene passata l'estensione. I nomi di file di più di 8 caratteri vengono troncati. Se un file ha lo stesso nome di un file già esistente sul disco, verrà scritto sovrapposto al primo. Se non viene specificato il dispositivo, viene utilizzato il drive di default.
L'opzione A salva il programma in formato ASCII, altrimenti il BASIC lo salva in formato binario compresso. I file ASCII occupano più spazio, ma alcuni tipi di accesso richiedono esplicitamente che il file sia in formato ASCII. Ad esempio, un file da fondere deve essere salvato in formato ASCII; i file salvati in formato ASCII possono infine essere letti anche come file di dati.
L'opzione P salva il programma in formato binario codificato: questa è un'opzione di protezione. Quando un programma protetto viene successivamente eseguito o caricato in memoria, ogni tentativo di eseguire LIST o EDIT fallirà, con la visualizzazione del messaggio d'errore ILLEGAL FUNCTION CALL.

NOTA: il nome del file presente nel catalogo del disco non dà indicazioni sul fatto se il file sia stato salvato protetto o in formato ASCII. Viene usata sempre l'estensione .BAS.

## SCREEN (ALT S) Funzione

Restituisce il codice ASCII (da 0 a 255) per il carattere posto sullo schermo alla riga e colonna specificate.

*Sintassi:*
*varnum* = SCREEN(*riga,colonna* [,*z*])

*Esempio:*

```
PRINT SCREEN (M+20,1,C)
D=SCREEN (1,1)
```

Il parametro *riga* è un'espressione numerica nell'intervallo tra 1 e 25; *colonna* è un'espressione numerica nell'intervallo tra 1 e 40 o tra 1 e 80, secondo quanto definito da WIDTH; *z* infine è un'espressione numerica che verifica se una condizione sia vera o falsa. Il parametro *z* è utilizzabile solo in modo Text.
In modo Text, se *z* è indicato e vero (diverso da zero), il colore attribuito per il carattere viene restituito al posto del codice del carattere. Il colore attribuito è un numero dell'intervallo da 0 a 255: questo numero, *varnum*, può essere decifrato nel seguente modo:

— (*varnum* MOD 16) è il colore del testo
— ((*varnum* MOD 128) \ 16) è il colore del fondo, dove testo è calcolato come sopra
— (*varnum*>127) è vera (−1) se il carattere è lampeggiante, falsa (0) se non lo è.

Vedere COLOR per l'elenco dei colori e dei numeri loro associati.
In modo Grafico, se la locazione specificata contiene informazioni grafiche (punti o linee, contrapposte ai soli caratteri), allora la funzione SCREEN restituisce zero. Ogni valore al di fuori dell'intervallo indicato causerà un errore ILLEGAL FUNCTION CALL. Il seguente esempio è dimostrativo della funzione SCREEN:

```
100 X=SCREEN (10,10)
```

Se il carattere alla posizione (10,10) è A, allora X=65.
Il seguente esempio restituisce il colore attribuito al carattere nell'angolo in alto a sinistra dello schermo:

```
100 X=SCREEN (1,1,1)
```

## SCREEN (ALT S) Istruzione

Definisce gli attributi dello schermo che devono essere utilizzati dalle istruzioni successive. È significativa solo con la scheda Colore/Grafica.

*Sintassi:*
SCREEN [*modo*[,[*colore*] [,[*paginaa*] [,*paginav*]]]

*Esempi:*

```
SCREEN 1
SCREEN ,,4
SCREEN 0,0,0
SCREEN 1,0,3,1
```

Il parametro *modo* è un'espressione numerica che può assumere i valori 0, 1, o 2. I modi validi sono:

0 modo Text con la larghezza di schermo corrente (40 o 80).
1 modo grafico in media risoluzione (320×200). Utilizzabile solo con la scheda Colore/Grafica.
2 modo Grafico ad alta risoluzione (640×200). Utilizzabile solo con la scheda Colore/Grafica.

Il parametro *colore* è un'espressione che abilita o disabilita i colori, a seconda del fatto che abbia un valore vero o falso. In modo Text (modo=0) un valore falso (zero) disabilita il colore (ciò consente solo la visualizzazione di immagini in bianco e nero) mentre un valore vero (non zero) abilita i colori (permette di visualizzare immagini a colori). Nel modo Grafico a media risoluzione (modo=1) un valore vero disabilita il colore, mentre uno falso lo abilita. Poiché i colori bianco e nero sono gli unici ottenibili nel modo Grafico ad alta risoluzione (modo=2), in questo caso il parametro *colore* non avrà alcun effetto.
Il parametro *paginaa* (pagina attiva) è un'espressione intera nell'intervallo tra 0 e 7 per una larghezza di 40 caratteri, e nell'intervallo tra 0 e 3 per 80 che sceglie la pagina di schermo su cui devono comparire gli output inviati, ed è valida solo in modo Text.
Il parametro *paginav* (pagina visualizzata) sceglie la pagina da visualizzare sullo schermo, nello stesso modo della pagina attiva vista prima; le due pagine possono differire. *paginav* è valido solo in modo Text; se omesso, la pagina visualizzata è per default quella attiva.
Se tutti i parametri indicati sono validi, viene memorizzato il nuovo modo dello schermo, che viene cancellato; il colore del testo diviene il bianco, mentre il colore del fondo e del bordo è il nero. Nel caso in cui il nuovo modo fosse uguale al precedente, nulla verrebbe cambiato.

Se il modo è Text e vengono specificati solo *paginaa* e *paginav*, l'effetto è quello di cambiare la pagina visualizzata. Inizialmente sia la pagina visiva che quella attiva per default sono la pagina 0; modificando i valori dei parametri, potete visualizzare una pagina mentre ne state producendo un'altra e poi potete scambiarle immediatamente l'una con l'altra.

> NOTA: un solo cursore viene condiviso tra tutte le pagine; se continuate a modificare la pagina attiva, dovete mantenere il ricordo della posizione del cursore per la pagina attiva corrente (utilizzando POS (0) e CRSLIN) prima di passare ad un'altra pagina attiva; quando poi ritornate all'originale, potete riportare il cursore in posizione per mezzo dell'istruzione LOCATE.

Ciascuno dei parametri può essere omesso ed in tal caso (tranne *paginav*) assume il vecchio valore. Ogni valore al di fuori dell'intervallo indicato provoca un errore ILLEGAL FUNCTION CALL; i valori precedenti vengono così mantenuti.

## **SGN** Funzione

Restituisce il segno di *esprnum*.

*Sintassi:*
*varnum* = SGN(*esprnum*)

*Esempio:*

```
ON SGN(X)+2 GOTO 100,200,300
```

SGN *esprnum* è la funzione matematica segno:

— se *esprnum* è positivo, SGN(*esprnum*) vale 1
— se *esprnum* è zero, SGN(*esprnum*) vale 0
— se *esprnum* è negativo, SGN(*esprnum*) vale −1.

## **SIN** Funzione

Restituisce il seno di *angolo*.

*Sintassi:*
*varnum* = SIN(*angolo*)

*Esempio:*

```
PRINT SIN (D*pi/180)
```

Il parametro *angolo* è il valore dell'angolo in radianti: se volete convertire gradi in radianti, moltiplicate il vostro valore per $\pi/180$, dove $\pi=3.141593$. SIN (*angolo*) viene calcolato in precisione semplice, a meno che il BASIC non sia stato avviato con l'opzione /D.

## **SOUND** Istruzione

Genera suoni per mezzo dell'altoparlante.

*Sintassi:*
SOUND *freq, durata*

*Esempio:*

```
SOUND 440,5000
```

Il parametro *freq* indica la frequenza richiesta in Hertz (cicli al secondo). È un'espressione numerica compresa tra 37 e 32767; *durata* è la durata richiesta in "zic" del clock (uno zic equivale a 1/18,2 secondi) e può essere un'espressione numerica compresa tra 0 e 65535.
Quando l'istruzione SOUND produce un suono, il programma continua ad eseguirlo fino a quando non incontra un'altra istruzione SOUND: se *durata* della nuova istruzione SOUND è zero, viene interrotto il suono corrente, altrimenti il programma attende il termine della prima esecuzione prima di iniziare la seconda.
Se state utilizzando l'Advanced BASIC, potete inserire i suoni in un buffer, in modo che l'esecuzione non si interrompa ogni volta che viene trovata una nuova istruzione (vedi il comando MB spiegato in PLAY, per ulteriori dettagli). Se nessuna istruzione SOUND è in esecuzione, l'istruzione SOUND *freq*,0 non ha alcun effetto. La nota base è il La con frequenza 440 Hz; per le altre frequenze fate riferimento alla Tabella A.1.
Per creare dei periodi di silenzio utilizzate l'istruzione SOUND 32767, *durata*. La durata di un battito può essere calcolata dai battiti per minuto dividendoli per 1092 (il numero di zic in un minuto). Nella Tabella A.2 sono riportati i tempi caratteristici in funzione degli zic.

**Tabella A.1** Correlazione tra note e frequenze

| Nota | Frequenza | Nota | Frequenza |
|---|---|---|---|
| C (Do) | 130.810 | C* (Do) | 523.250 |
| D (Re) | 146.830 | D (Re) | 587.330 |
| E (Mi) | 164.810 | E (Mi) | 659.260 |
| F (Fa) | 174.610 | F (Fa) | 698.460 |
| G (Sol) | 196.000 | G (Sol) | 783.990 |
| A (La) | 220.000 | A (La) | 880.000 |
| B (Si) | 246.940 | B (Si) | 987.770 |
| C (Do) | 261.630 | C (Do) | 1046.500 |
| D (Re) | 293.660 | D (Re) | 1174.700 |
| E (Mi) | 329.630 | E (Mi) | 1318.500 |
| F (Fa) | 349.230 | F (Fa) | 1396.900 |
| G (Sol) | 392.000 | G (Sol) | 1568.000 |
| A (La) | 440.000 | A (La) | 1760.000 |
| B (Si) | 493.880 | B (Si) | 1975.500 |

*Do centrale. Le note più alte o più basse possono essere ottenute approssimativamente raddoppiando o dimezzando la frequenza della nota corrispondente nell'ottava precedente o successiva.

**Tabella A.2** Tempi caratteristici

| | Tempo | Battiti per minuto | Durata (in zic) del battito |
|---|---|---|---|
| Molto lento | Larghissimo | | |
| | Largo | 40-60 | 27.3-18.2 |
| | Larghetto | 60-66 | 18.2-16.55 |
| | Grave | | |
| | Lento | | |
| | Adagetto | 66-76 | 16.55-14.37 |
| Lento | Adagio | | |
| | Andante | 76-108 | 14.37-10.11 |
| Medio | Andantino | | |
| | Moderato | 108-120 | 10.11-9.1 |
| Veloce | Allegretto | | |
| | Allegro | 120-168 | 9.1-6.5 |
| | Vivace | | |
| | Veloce | | |
| | Presto | 168-208 | 6.5-5.25 |
| Molto veloce | Prestissimo | | |

## SPACE$ Funzione

Restituisce una stringa di *esprnum* spazi bianchi.

*Sintassi:*
*varstr* = SPACE$(*esprnum*)

*Esempio:*

```
X$=X$+SPACE$(5)
```

Il parametro *esprnum* deve appartenere all'intervallo compreso tra 0 e 255. Vedi anche la funzione SPC.

## SPC Funzione

Salta *esprnum* spazi in un'istruzione PRINT.

*Sintassi:*
PRINT SPC (*esprnum*)

*Esempio:*

```
PRINT SPC(n+1)
```

Il parametro *esprnum* dev'essere compreso tra 0 e 255: se è maggiore della larghezza definita per il dispositivo, il valore utilizzato è *esprnum* MOD *larghezza*. SPC può essere usato solo con le istruzioni PRINT, LPRINT e PRINT#.
Se una funzione SPC si trova alla fine dell'elenco dei termini, il BASIC non aggiunge il carattere CR, come se la funzione SPC contenesse un implicito punto e virgola.

## SQR Funzione

Restituisce la radice quadrata di *esprnum.*

*Sintassi:*
*varnum* = SQR (*esprnum*)

*Esempio:*

```
D=SQR(B^2-4*A*C)
```

Il parametro *esprnum* deve essere maggiore o uguale a zero. SQR viene calcolata in precisione semplice a meno che il BASIC non sia stato avviato con l'opzione /D.

## **STICK** Funzione

Restituisce le coordinate x e y di due joystick.

*Sintassi:*
*varnum* = STICK(*esprnum*)

*Esempio:*

```
Q=STICK(0)
```

Il parametro *esprnum* dev'essere compreso tra 0 e 3 e determina il risultato nel modo seguente:

0 Restituisce la coordinata x del joystick A
1 Restituisce la coordinata y del joystick A
2 Restituisce la coordinata x del joystick B
3 Restituisce la coordinata y del joystick B

STICK(0) calcola tutti e quattro i valori delle coordinate ma restituisce solo il valore di STICK(0); STICK(1), STICK(2) e STICK(3) invece non controllano il joystick, ma restituiscono semplicemente i valori precedentemente calcolati da STICK(0).
L'intervallo di valori per x ed y dipende dal joystick utilizzato.

## **STOP** Istruzione

Termina l'esecuzione del programma e ritorna al livello comandi.

*Sintassi:*
STOP

*Esempio:*

```
STOP
```

L'istruzione STOP può essere utilizzata in qualunque punto di un programma per farne terminare l'esecuzione. Quando il BASIC trova un'istruzione STOP, visualizza il seguente messaggio:

```
Break in nnnnn
```

ove *nnnnn* è il numero di linea in cui si trovava l'istruzione STOP. Diversamente dall'istruzione END, STOP non chiude i file; il BASIC torna sempre al livello comandi dopo l'esecuzione di un'istruzione STOP. Potete far riprendere l'esecuzione inviando il comando CONT.

## STR$ Funzione

Restituisce una rappresentazione in formato stringa del valore di *esprnum.*

*Sintassi:*
*varstr* = STR$(*esprnum*)

*Esempio:*

```
A$=STR$(A)
```

Se *esprnum* è positivo, la stringa prodotta da STR$ contiene uno spazio vuoto all'inizio (lo spazio riservato per il segno più). Per esempio:

```
Ok
?STR$(321); LEN(STR$(321))
 321 4
Ok
```

La funzione VAL è complementare di STR$.

## STRIG Funzione

Restituisce lo stato dei pulsanti del joystick.

*Sintassi:*
*varnum* = STRIG(*esprnum*)

*Esempio:*

```
IF STRIG(S) THEN GOSUB 9000
```

Il parametro *esprnum* dev'essere compreso tra 0 e 3 e determina i valori restituiti nel modo seguente:

0 Restituisce −1 se il pulsante A1 è stato premuto dopo l'ultima chiamata della funzione STRIG(0), 0 in caso contrario.
1 Restituisce −1 se il pulsante A1 è premuto, 0 in caso contrario.
2 Restituisce −1 se il pulsante B1 è stato premuto dopo l'ultima chiamata della funzione STRIG(2), 0 in caso contrario
3 Restituisce −1 se viene premuto il pulsante B1, 0 in caso contrario.

In Advanced BASIC e in Compiled BASIC potete leggere quattro pulsanti del joystick. Gli ulteriori valori di *esprnum* sono:

4 Restituisce −1 se il pulsante A2 è stato premuto dopo l'ultima chiamata della funzione STRIG(4), 0 in caso contrario.
5 Restituisce −1 se viene premuto il pulsante A2, 0 in caso contrario.
6 Restituisce −1 se il pulsante B2 è stato premuto dopo l'ultima chiamata della funzione STRIG(6), 0 in caso contrario.
7 Restituisce −1 se viene premuto il pulsante B2, 0 in caso contrario.

Prima di poter chiamare la funzione STRIG(*esprnum*) è necessario eseguire un'istruzione STRIG ON; dopo di questa, ogni volta che il programma inizia a considerare una nuova istruzione, il BASIC controlla se è stato premuto un pulsante del joystick. Invece, se STRIG è OFF, non avviene alcun controllo.

## **STRIG** Istruzione

Abilita o disabilita i pulsanti del joystick.

*Sintassi:*
STRIG ON
STRIG OFF

## **STRIG (n)** Istruzione

Abilita e disabilita l'intercettamento dei pulsanti del joystick. È valida solo in Advanced BASIC ed in Compiled BASIC.

*Sintassi:*
STRIG(*pulsante*) ON
STRIG(*pulsante*) OFF
STRIG(*pulsante*) STOP

*Esempi:*

```
STRIG(0) ON
IF ST=-1 THEN STRIG(4) STOP
```

Il parametro *pulsante* può valere 0, 2, 4, o 6 ed indica il pulsante da intercettare, nel modo seguente:

0 pulsante A1
2 pulsante B1
4 pulsante A2
6 pulsante B2

L'istruzione STRIG(*pulsante*) ON dev'essere eseguita per abilitare l'intercettamento grazie all'istruzione ON STRIG(*pulsante*). Dopo l'istruzione STRIG(*pulsante*) ON, ogni volta che il programma esegue una nuova istruzione il BASIC controlla se è stato premuto il pulsante indicato.
Se viene eseguita STRIG(*pulsante*) OFF, non avviene alcun controllo né alcun intercettamento: anche se viene premuto l'evento, non viene ricordato.
Se invece viene eseguita l'istruzione STRIG(*pulsante*) STOP, non ha luogo l'intercettamento, ma ogni evento viene ricordato per far scattare l'intercettamento non appena venga eseguita un'istruzione STRIG(*pulsante*) ON.

## **STRING$** Funzione

Restituisce una stringa, di lunghezza definibile, i cui caratteri hanno come codice ASCII il *codice* o il primo carattere di *stringa*.

*Sintassi:*
*varstr* = STRING$(*lunghezza, codice*)
*varstr* = STRING$(*lunghezza, stringa*)

*Esempio:*

```
PRINT STRIG$(40,45)
PRINT STRIG$(5,"A")
```

I parametri *codice* e *lunghezza* devono essere compresi tra 0 e 255.

## **SWAP** Istruzione

Scambia i valori di due variabili.

*Sintassi:*
SWAP *variabile1, variabile2*

*Esempio:*

```
SWAP A$,B$
```

I parametri *variabile1* e *variabile2* sono i nomi di due variabili o elementi di un array. Ogni tipo di variabile può essere scambiato (intero, precisione semplice, doppia precisione, stringa) purché le due variabili siano dello stesso tipo, altrimenti si incorre nell'errore TYPE MISMATCH.

## **SYSTEM** Comando

Esce dal BASIC e ritorna al DOS.

*Sintassi:*
SYSTEM

*Esempio:*

```
SYSTEM
```

Il comando SYSTEM chiude tutti i file prima di tornare al DOS e il vostro programma in BASIC viene perso. Se siete entrati nel BASIC per mezzo di un file batch dal DOS, il comando SYSTEM vi riporta al file batch, che continua ad essere eseguito nel punto in cui era stato interrotto.

## **TAB** Funzione

Esegue la tabulazione indicata da *esprnum.*

*Sintassi:*
PRINT TAB(*esprnum*)

*Esempio:*

```
PRINT TAB(Q);"Intestazione"
```

Il parametro *esprnum* dev'essere contenuto nell'intervallo che va da a 1 a 255: se la posizione corrente di scrittura è già al di là del punto indicato da *esprnum,* TAB si posiziona nella riga successiva, in *esprnum.* Il punto

1 è la posizione più a sinistra, mentre quella più a destra dipende dal valore definito nel comando WIDTH. TAB può essere usato solo nelle istruzioni PRINT, LPRINT e PRINT#.
Se la funzione TAB si trova alla fine di un elenco di termini, il BASIC non aggiunge il ritorno carrello, come se TAB contenesse un punto e virgola implicito.

## **TAN** Funzione

Restituisce la tangente di *angolo*.

*Sintassi:*
*varnum* = TAN(*angolo*)

*Esempio:*

```
PRINT TAN(D)
```

Il parametro *angolo* indica il valore dell'angolo in radianti: per convertire gradi in radianti, bisogna moltiplicarli per $\pi/180$, dove $\pi=3.141593$. TAN(*angolo*) viene calcolata in precisione semplice, tranne nel caso in cui il BASIC sia stato caricato con l'opzione /D.

## **TIME$** Istruzione

Definisce l'ora.

*Sintassi:*
TIME$ = "*stringa*"

*Esempio:*

```
TIME$="0:23:30"
```

*stringa* può essere inserita in uno di questi modi:

*hh*
Definisce l'ora nell'intervallo da 0 a 23. I minuti ed i secondi vengono posti per default a zero.

*hh:mm*
Definisce ore e minuti. I minuti devono essere compresi tra 0 e 59; i secondi per default vengono posti uguali a zero.

*hh:mm:ss*
Definisce ore, minuti e secondi. Anche i secondi devono essere compresi tra 0 e 59.

Lo zero che precede un numero di una sola cifra può essere omesso in ogni termine, ma dovete indicare sempre almeno una cifra. Se volete inserire come ora mezzanotte e trenta, potete battere TIME$="0:30", ma non TIME$=":30". Se uno qualsiasi dei valori è al di fuori dell'intervallo consentito, viene visualizzato il messaggio d'errore ILLEGAL FUNCTION CALL e viene mantenuta l'ora inserita precedentemente. Se *stringa* non è una stringa valida, invece, l'errore è TYPE MISMATCH.

## **TIME$** Variabile

Contiene l'ora corrente.

*Sintassi:*
*varstr* = TIME$

*Esempio:*

```
V$=TIME$
```

L'ora corrente viene restituita come una stringa di otto caratteri, della forma *hh:mm:ss*, ove *hh* è l'ora (da 00 a 23), *mm* i minuti (da 00 a 59) ed *ss* i secondi (da 00 a 59). L'ora può essere stata definita nel DOS prima di entrare in BASIC.

## **TIMER** Funzione

Restituisce un valore in precisione semplice che rappresenta il numero di secondi trascorsi dalla mezzanotte o dall'ultima reinizializzazione del sistema. È valida solo in Disk BASIC ed in Advanced BASIC.

*Sintassi:*
*varnum* = TIMER

*Esempio:*

```
SEC_CNT=TIMER
```

Il valore restituito dalla funzione TIMER calcola i secondi trascorsi

dall'avviamento del sistema (cifre decimali). TIMER è una funzione "a sola lettura".

## **TRON e TROFF** Comandi

Visualizzano passo passo l'esecuzione del programma.

*Sintassi:*
TRON
TROFF

*Esempi:*

```
TRON
TROFF
```

Come aiuto nel debugging, il comando TRON (che può anche essere utilizzato in modo indiretto) abilita un flag di "traccia" che visualizza ogni numero di linea corrispondente alla linea di programma che viene eseguita. Il numero appare racchiuso tra parentesi quadre. Il tracciamento viene disabilitato dal comando TROFF.

## **USR** Funzione

Richiama la subroutine indicata in linguaggio macchina, con argomento *arg*.

*Sintassi:*
*varnum* = USR [*n*] (*arg*)

*Esempio:*

```
PRINT USR5(13)
Q = USR(4)
```

Il parametro *n*, nell'intervallo tra 0 e 9 corrisponde al valore fornito nell'istruzione DEF USR per la routine desiderata. Se *n* viene omesso, si assume USR 0; *arg* è una qualsiasi espressione numerica o variabile a stringa che diverrà l'argomento della subroutine in linguaggio macchina. L'istruzione CALL è un modo alternativo per richiamare una subroutine in linguaggio macchina.

## **VAL** (ALT V) Funzione

Restituisce il valore numerico di *stringa*.

*Sintassi:*
*varnum* = VAL (*stringa*)

*Esempio:*

```
A=VAL(A$)
```

La funzione VAL toglie spazi bianchi, tabulazioni e a capo dalla *stringa* argomento per determinare il risultato. Per esempio:

```
VAL (" -3")
```

restituisce il valore −3. Se il primo carattere di stringa non è un carattere numerico, VAL (*stringa*) restituirà il valore 0. Vedere la funzione STR$ per la conversione da numeri in stringhe.

## **VARPTR** Funzione

Restituisce l'indirizzo di memoria della variabile o il blocco di controllo del file.

*Sintassi:*
*varnum* = VARPTR(*variabile*)
*varnum* = VARPTR(# *numfile*)

*Esempio:*

```
Q%=VARPTR (N$)
Q%=VARPTR (#2)
```

Il parametro *variabile* è il nome di una variabile a stringa o numerica o di un elemento di un array del programma. A *variabile* dev'essere assegnato un valore prima di chiamare VARPTR altrimenti si verificherà un errore ILLEGAL FUNCTION CALL; *numfile* è lo stesso numero con cui è stato aperto il file.
Per entrambe le sintassi, l'indirizzo restituito è un numero intero nell'intervallo tra 0 e 65535, che indica l'offset nel segmento dei dati BASIC. Questo indirizzo non viene influenzato dall'istruzione DEF SEG. La pri-

ma sintassi restituisce l'indirizzo del primo byte del dato identificato da *variabile*.

NOTA: a tutte le variabili semplici deve essere assegnato un valore prima di richiamare VARPTR per un array, poiché gli indirizzi degli array cambiano ogni volta che viene utilizzata una nuova variabile semplice.

VARPTR è normalmente utilizzata per ottenere l'indirizzo di una variabile o di un array in modo da poterlo passare ad una routine USR in linguaggio macchina. Una chiamata di funzione della forma VARPTR(A(0)) è normalmente indicata quando si passa un array in modo che sia restituito l'elemento dell'array con indirizzo più basso.
La seconda sintassi restituisce l'indirizzo di partenza del blocco di controllo del file indicato. Questo non corrisponde al blocco DOS di controllo del file.

## **VARPTR$** Funzione

Restituisce il valore dell'indirizzo di una variabile in memoria in un formato carattere. È utilizzata soprattutto con PLAY e DRAW in programmi che verranno poi compilati.

*Sintassi:*
*varstr* = VARPTR$(*variabile*)

*Esempio:*

```
N$=VARPTR$ (P)
```

Il parametro *variabile* è il nome di una variabile esistente nel programma.

NOTA: a tutte le variabili semplici deve essere assegnato un valore prima di richiamare VARPTR per un array, poiché gli indirizzi degli array cambiano ogni volta che viene utilizzata una nuova variabile semplice.

VARPTR$ restituisce una stringa di tre byte nella forma seguente:

| **Byte 0** | **Byte 1** | **Byte 2** |
|---|---|---|
| *tipo* | Byte più basso dell'indirizzo | Byte più alto dell'indirizzo |

dove *tipo* indica il tipo di variabile:

2 Intero
3 Stringa
4 Precisione semplice
8 Doppia precisione

Il valore restituito è lo stesso di

CHR$(*tipo*)+MKI$(VARPTR(*variabile*))

Potete usare VARPTR$ per indicare il nome di variabile nella stringa di comando dell'istruzione PLAY o DRAW, come in questo esempio:

| **Metodo A** | **Metodo B** |
|---|---|
| PLAY "XA$;" | PLAY "X"+VARPTR$(A$) |
| PLAY "0=1;" | PLAY "0="+VARPTR$(1) |

Questa tecnica è utilizzata soprattutto nei programmi che verranno compilati in seguito.

## **VIEW** Istruzione

Permette di suddividere la superficie visibile (lo schermo) in porzioni dette viewport. Il contenuto della finestra verrà poi visualizzato nella viewport corrente. È un'istruzione valida solo in Advanced BASIC ed in modo Grafico.

*Sintassi:*
VIEW [[SCREEN] [(*x1*,*y1*)–(*x2*,*y2*) [,[*colore*] [,[*bordo*]]]]]

*Esempi:*

```
VIEW (10,10)-(110,100)
VIEW SCREEN (5,5)-(200,100),,2,3
```

I parametri (*x1*,*y1*) e (*x2*,*y2*) definiscono gli angoli opposti del rettangolo che forma la viewport. I valori di *x* e di *y* devono essere compresi dai limiti effettivi dello schermo, altrimenti si verifica un errore ILLEGAL FUNCTION CALL. VIEW ordina le coppie di argomenti *x* ed *y* e pone per primi i valori più bassi di *x* e di *y*. Tutti i possibili accoppiamenti di *x* e di *y* sono validi (all'interno dei limiti dello schermo) purché *x1* non sia uguale a *x2* ed *y1* non sia uguale ad *y2*.

Il parametro *colore* indica il colore con cui la viewport sarà riempita (colore di fondo); se viene omesso, essa non viene riempita e tutti gli oggetti già presenti all'interno di quell'area restano immutati. Il parametro *bordo* definisce il colore della linea con cui verrà tracciato il contorno dell'area della viewport: se viene omesso, non viene disegnato alcun contorno. Sia *colore* che *bordo* possono variare da 0 a 3 in media risoluzione e scelgono il colore della tavolozza corrente. In alta risoluzione i parametri possono essere 0 (nero) o 1 (bianco).
Se l'argomento SCREEN è omesso, tutti i punti disegnati hanno coordinate relative alle viewport, cioè *x1* ed *y1* vengono aggiunti alle coordinate *x* ed *y* prima di disegnare il punto sullo schermo; se viene aggiunto SCREEN, tutti i punti disegnati hanno coordinate assolute e possono cadere all'interno o all'esterno dei limiti della viewport. In ogni caso risultano visibili solo quei punti che cadono all'interno; i punti al di là dei confini della viewport vengono eliminati.
Se l'istruzione VIEW non ha argomenti, la viewport risultante sarà l'intera superficie visibile dello schermo; in altre parole, per la media risoluzione equivale a VIEW (0,0)−(319,199), mentre per l'alta risoluzione a VIEW (0,0)−(639,199).
Per mezzo dell'istruzione VIEW possono essere definite numerose viewport, ma sarà in vigore solo la più recente. I comandi RUN e SCREEN disabilitano ogni viewport eventualmente attiva e definiscono l'intero schermo come viewport corrente.
Potete dare una nuova scala ad un oggetto cambiando le dimensioni della viewport. Una viewport più grande renderà l'oggetto più grande, mentre una più piccola lo farà apparire più piccolo: questo è simile all'ingrandimento ed alla riduzione ottenuti con l'istruzione WINDOW, ma la dimensione degli oggetti cambia solo per le istruzioni eseguite dopo l'istruzione VIEW. Vedere l'istruzione WINDOW per maggiori informazioni.

> NOTA: l'istruzione CLS agisce solo sulla viewport corrente; se perciò volete cancellare l'intero schermo quando ne state utilizzando solo una parte come viewport, dovete usare prima VIEW per tornare a tutto lo schermo e poi usare il comando CLS. Utilizzando CLS con una viewport il cursore non viene riportato nella posizione home. Potete usare i tasti CTRL HOME per cancellare l'intero schermo e riportare il cursore nella posizione home.

## **WAIT** Istruzione

Sospende l'esecuzione del programma mentre controlla lo stato di una porta di input.

*Sintassi:*
WAIT *porta, and*[,*or*]

*Esempio:*

```
WAIT 34,2
```

Il parametro *porta* è un numero compreso tra 0 e 65535; *and* e *or* sono espressioni intere nell'intervallo da 0 a 255; fate riferimento al manuale *Technical Reference* per una descrizione dei numeri validi per le porte (indirizzi di I/O).
L'istruzione WAIT fa sospendere l'esecuzione finché una specifica porta di input non riceve una specifica configurazione di bit.
Il dato è sottoposto a uno XOR con l'espressione intera *or* e poi ad un AND con l'espressione *and*. Se il risultato è zero, il BASIC torna indietro e legge di nuovo il dato che compare nella porta, altrimenti l'esecuzione continua con l'istruzione successiva. Se *or* viene omessa, viene assunta uguale a 0.
L'istruzione WAIT vi permette di controllare uno o più bit in una porta di input: potete controllare se è un 1 o uno 0. Le posizioni da controllare sono definite ponendo un 1 in quella posizione nell'espressione *and*. Se non indicate *or*, i bit della porta di input vengono confrontati con gli 1, altrimenti, se lo indicate, un 1 in una qualsiasi posizione di *or* (per la quale ci sia un 1 in *and*) farà sì che WAIT confronti il bit in input con uno zero.
Durante l'esecuzione, l'istruzione WAIT compie un loop continuo, controllando gli input corrispondenti agli 1 in *and*. Se uno di questi bit vale 1 (o zero, se il bit corrispondente in *or* vale 1) il programma continua dall'istruzione seguente. In questo modo WAIT non attende che sia riconosciuta tutta la configurazione dei bit, ma si accontenta del primo che appare.

NOTA: è possibile entrare in un loop infinito usando l'istruzione WAIT; per uscire dal loop potete usare i tasti CTRL BREAK o una reinizializzazione del sistema.

## **WHILE - WEND** Istruzione

Esegue una serie di istruzioni in un loop che dura finché la condizione indicata è vera.

*Sintassi:*

WHILE *espressione*
:
(*istruzioni del loop*)
:
WEND

*Esempio:*

```
WHILE IN
  .
  .
  .
WEND
```

Se *espressione* è vera (non zero) le *istruzioni del loop* vengono eseguite fino all'istruzione WEND; il BASIC a questo punto ritorna all'istruzione WHILE e controlla *espressione*. Se è ancora vera, il processo viene ripetuto, se invece non è vera, l'esecuzione riprende dall'istruzione che segue WEND.
I loop WHILE-WEND possono essere annidati a più livelli; ogni istruzione WEND corrisponde all'istruzione WHILE più recente. Un'istruzione WHILE non corrisposta produce un errore WHILE WITHOUT WEND, mentre un'istruzione WEND non corrisposta produce l'errore WEND WITHOUT WHILE.

## **WIDTH** (ALT W) Istruzione

Definisce il numero di caratteri che compongono la larghezza della linea di output; dopo il numero indicato, il BASIC aggiunge un ritorno carrello.

*Sintassi:*

WIDTH *dimensione*
WIDTH *numfile, dimensione*
WIDTH *"dispositivo", dimensione*

*Esempi:*

```
WIDTH 80
WIDTH #2,32
WIDTH "LPT2:",132
```

Il parametro *dimensione* è un'espressione numerica compresa tra 0 e 255 che definisce la nuova larghezza. WIDTH 0 equivale a WIDTH 1; *numfile*

è un'espressione numerica nell'intervallo da 1 a 15 che corrisponde al numero del file aperto in uno dei dispositivi sotto elencati; *dispositivo* è un'espressione di tipo stringa per l'identificatore del dispositivo. I dispositivi validi sono: SCRN:, LPT1:, LPT2:, LPT3:, COM1: e COM2:.
A seconda del dispositivo indicato, sono possibili le seguenti azioni:

WIDTH *dimensione* o WIDTH "SCRN:",*dimensione*

Definisce la larghezza dello schermo. Sono permesse solo combinazioni con 40 o 80 colonne.
Se lo schermo è in modo Grafico a media risoluzione (come accade con un'istruzione SCREEN 1), una larghezza di 80 caratteri (WIDTH 80) fa passare lo schermo in alta risoluzione, cioè equivale ad un'istruzione SCREEN 2.
Se lo schermo è in modo Grafico ad alta risoluzione (istruzione SCREEN 2), l'istruzione WIDTH 40 lo riporta in media risoluzione (come l'istruzione SCREEN 1).
NOTA: cambiare la larghezza dello schermo, fa sì che questo venga cancellato e che il bordo diventi nero.

WIDTH *"dispositivo",dimensione*

Usata per assegnare un'appropriata larghezza di stampa ad un dispositivo. Questa sintassi di WIDTH memorizza il nuovo valore della larghezza senza variare quello corrente; una successiva istruzione OPEN del dispositivo utilizzerà questo valore per la larghezza del file aperto. La larghezza però non cambia subito se il file è già aperto.
NOTA: LPRINT, LLIST e LIST, "LPT*n*:" effettuano implicitamente un'istruzione OPEN e vengono perciò influenzate da questa istruzione.

WIDTH *numfile,dimensione*

La larghezza del dispositivo associato a *numfile* è immediatamente modificata quando viene specificato il nuovo valore; si può cambiare così la larghezza quando il file è già aperto. Questa forma agisce su LPT1:, LPT2:, LPT3:, COM1:, COM2:.

Ogni valore all'esterno degli intervalli indicati produce un errore ILLEGAL FUNCTION CALL e viene mantenuto il valore precedente. WIDTH non ha effetto per la tastiera (KYBD:).
La larghezza di tutte le stampanti è per default 80 caratteri; il massimo per la stampante IBM è 132 caratteri, anche se non vengono visualizzati messaggi d'errore per valori tra 132 e 255.
È compito vostro definire la larghezza fisica che più vi serve sulla stampante. Alcune stampanti vengono predisposte inviando codici speciali, mentre altre hanno alcuni interruttori da sistemare. Per la stampante IBM dovete utilizzare LPRINT CHR$(15); per passare ad una stampa condensata quando utilizzate larghezze maggiori di 80.

Per ritornare alla stampa normale usate LPRINT CHR$(18);. Questa stampante è già predisposta per aggiungere un carattere CR automaticamente se il numero di caratteri oltrepassa quello definito per la linea di stampa. Indicando una lunghezza di 255 caratteri si elimina la suddivisione in righe, cioè si ha l'effetto di una "linea infinita": questo è il valore di default per i file di comunicazione. Cambiando la lunghezza per i file di comunicazione non si altera né il buffer di trasmissione né quello di ricezione, ma semplicemente si comanda al BASIC di inviare un carattere di ritorno carrello dopo ogni *dimensione* caratteri.
Il cambiamento del modo dello schermo modifica la larghezza dello schermo solo quando si passa da SCREEN 2 a SCREEN 1 o SCREEN 0.

## **WINDOW** Istruzione

Permette di ridefinire le coordinate dello schermo. Valida solo in Advanced BASIC e in modo Grafico.

*Sintassi:*

WINDOW [[SCREEN] (*x1,y1*)−(*x2,y2*)]

*Esempio:*

```
WINDOW SCREEN (-1,-1)-(1,1)
```

Le coordinate (*x1,y1*) e (*x2,y2*) definiscono lo spazio delle coordinate esterne che verrà poi riportato nello spazio delle coordinate fisiche. (Lo spazio delle coordinate fisiche può essere ridefinito utilizzando l'istruzione VIEW). Le coordinate (*x1,y1*) e (*x2,y2*) sono numeri in virgola mobile ed in precisione semplice che definiscono gli angoli opposti dell'area rettangolare scelta come finestra.
Dopo aver definito la finestra, gli oggetti che volete disegnare utilizzeranno le coordinate esterne, invece che i limiti logici dello schermo. Il BASIC esegue la conversione delle coordinate esterne nelle appropriate coordinate fisiche dello schermo. L'istruzione WINDOW libera il programmatore dai problemi della trasformazione delle coordinate reali (esterne) in modo che rispettino le limitazioni fisiche dello schermo.
Se non viene inserito l'attributo SCREEN, lo schermo è considerato come un quadro di assi cartesiani, con l'angolo in basso a sinistra dello spazio esterno corrispondente al punto con i più bassi valori di *x* ed *y*; se invece viene aggiunto anche SCREEN, il punto con le coordinate *x* e *y* più basse si trova nell'angolo in alto a sinistra.
Tutti i possibili accoppiamenti di *x* ed *y* sono validi per la definizione della finestra, purché *x1* non sia uguale a *x2* ed *y1* non sia uguale ad *y2*.

WINDOW utilizza il taglio delle linee in modo che i punti che si trovano all'esterno dell'intervallo di variabilità delle coordinate non siano visibili all'interno dell'area visualizzata. Tutti gli oggetti che giacciono parzialmente all'esterno dell'intervallo delle coordinate vengono tagliati in modo che solo la parte interna sia visibile.
WINDOW permette di eseguire ingrandimenti e riduzioni degli oggetti, ridisegnandoli ogni volta dopo l'istruzione WINDOW. Se utilizzate una finestra con coordinate maggiori dell'oggetto, otterrete il disegno di tutto l'oggetto, ma l'immagine sarà piccola e circondata da spazi bianchi. Se scegliete coordinate più piccole, richiedete l'intervento del taglio delle linee, in modo da visualizzare solo un particolare, ingrandito, dell'oggetto stesso.
Se indicate coordinate sempre più grandi o più piccole potete ingrandire un oggetto fino a fargli occupare l'intero schermo (o ancora di più fino a distinguere in dettaglio anche particolari trascurabili), oppure potete rimpicciolirlo finché non resti più che un punto sullo schermo.
I comandi RUN, SCREEN e WINDOW senza attributi disabilitano ogni coordinata WINDOW precedentemente definita e riportano lo schermo in coordinate fisiche (in altre parole, il sistema di coordinate esterno viene posto uguale a quello delle coordinate fisiche).

## **WRITE** Istruzione

Visualizza dati sullo schermo.

*Sintassi:*
WRITE [*lista di espressioni*]

*Esempio:*

```
WRITE A,B,C$
```

Il parametro *lista di espressioni* è un elenco di espressioni numeriche o di tipo stringa o entrambe, separate da virgole o da punti e virgola; se *lista di espressioni* viene omessa, viene visualizzata una linea vuota, se invece è inserita nel comando, i valori delle espressioni compaiono sullo schermo.
Nella visualizzazione ogni valore viene separato dal successivo per mezzo di una virgola; le stringhe sono delimitate da virgolette. Dopo l'ultimo termine il BASIC aggiunge una sequenza CR/LF.
L'istruzione WRITE è simile a PRINT; la differenza tra le due è data dal fatto che WRITE inserisce una virgola tra i termini nella visualizzazione e delimita le stringhe con virgolette; inoltre i numeri positivi non sono preceduti da uno spazio bianco.

## **WRITE#** Istruzione

Scrive dati in un file sequenziale.

*Sintassi:*
WRITE# *numfile, lista di espressioni*

*Esempio:*

```
WRITE #2, A,B,C$
```

Il parametro *numfile* è il numero con cui il file era stato aperto per l'output; *lista di espressioni* è un elenco di espressioni numeriche o di tipo stringa o entrambe, separate da virgole o da punti e virgola.
La differenza tra WRITE# e PRINT# è data dal fatto che WRITE# inserisce una virgola tra i termini che vengono scritti e delimita le stringhe con virgolette; non è perciò necessario inserire delimitatori espliciti nella lista. Inoltre i numeri positivi non sono preceduti da uno spazio bianco. Dopo che l'ultimo termine della lista è stato scritto viene aggiunta una sequenza CR/LF.

Appendice

# B Codici dei caratteri

La tabella B.1 contiene i codici ASCII corrispondenti ai 256 caratteri del PC; questi codici possono essere utilizzati con la funzione CHR$ e con il tasto ALT per generare i caratteri sullo schermo. Vengono inoltre restituiti come valori numerici in risposta della funzione ASC.
I codici di alcuni tasti hanno un significato speciale per il PC; per distinguere questi codici dai 256 corrispondenti ai caratteri ASCII, il PC assegna loro una lunghezza di due byte: il primo byte di questo codice esteso è sempre NUL (codice 0), mentre il secondo byte identifica il tasto. Per questo la variabile di sistema INKEY$ restituisce un codice di due byte quando vengono premuti alcuni tasti. I valori ASCII elencati nella tabella B.2 corrispondono ai valori del secondo byte dei codici estesi: il tasto o i tasti che generano il codice sono riportati nella colonna "Significato".

**Tabella B.1** Codici ASCII per il PC

| Valore decimale | Valore esadecimale | Carattere di controllo | Carattere |
|---|---|---|---|
| 0 | 00 | NUL | Carattere nullo |
| 1 | 01 | SOH | ☺ |
| 2 | 02 | STX | ☻ |
| 3 | 03 | ETX | ♥ |
| 4 | 04 | EOT | ♦ |
| 5 | 05 | ENQ | ♣ |
| 6 | 06 | ACK | ♠ |
| 7 | 07 | BEL | Beep |
| 8 | 08 | BS | ◘ |
| 9 | 09 | HT | Tabulazione |
| 10 | 0A | LF | Line-feed |
| 11 | 0B | VT | Cursor home |
| 12 | 0C | FF | Form-feed |
| 13 | 0D | CR | Enter |
| 14 | 0E | SO | ♫ |
| 15 | 0F | SI | ☼ |
| 16 | 10 | DLE | ► |
| 17 | 11 | DC1 | ◄ |
| 18 | 12 | DC2 | ↕ |
| 19 | 13 | DC3 | ‼ |
| 20 | 14 | DC4 | ¶ |
| 21 | 15 | NAK | § |
| 22 | 16 | SYN | ▬ |
| 23 | 17 | ETB | ↨ |
| 24 | 18 | CAN | ↑ |
| 25 | 19 | EM | ↓ |
| 26 | 1A | SUB | → |
| 27 | 1B | ESC | ← |
| 28 | 1C | FS | Cursore a destra |
| 29 | 1D | GS | Cursore a sinistra |
| 30 | 1E | RS | Cursore in alto |
| 31 | 1F | US | Cursore in basso |
| 32 | 20 | | Spazio vuoto |
| 33 | 21 | | ! |
| 34 | 22 | | " |

**Tabella B.1** Codici ASCII per il PC (continua)

| Valore decimale | Valore esadecimale | Carattere di controllo | Carattere |
|---|---|---|---|
| 35 | 23 | | # |
| 36 | 24 | | $ |
| 37 | 25 | | % |
| 38 | 26 | | & |
| 39 | 27 | | ’ |
| 40 | 28 | | ( |
| 41 | 29 | | ) |
| 42 | 2A | | * |
| 43 | 2B | | + |
| 44 | 2C | | , |
| 45 | 2D | | - |
| 46 | 2E | | . |
| 47 | 2F | | / |
| 48 | 30 | | 0 |
| 49 | 31 | | 1 |
| 50 | 32 | | 2 |
| 51 | 33 | | 3 |
| 52 | 34 | | 4 |
| 53 | 35 | | 5 |
| 54 | 36 | | 6 |
| 55 | 37 | | 7 |
| 56 | 38 | | 8 |
| 57 | 39 | | 9 |
| 58 | 3A | | : |
| 59 | 3B | | ; |
| 60 | 3C | | < |
| 61 | 3D | | = |
| 62 | 3E | | > |
| 63 | 3F | | ? |
| 64 | 40 | | @ |
| 65 | 41 | | A |
| 66 | 42 | | B |
| 67 | 43 | | C |
| 68 | 44 | | D |
| 69 | 45 | | E |

**Tabella B.1** Codici ASCII per il PC (continua)

| Valore decimale | Valore esadecimale | Carattere di controllo | Carattere |
|---|---|---|---|
| 70 | 46 | | F |
| 71 | 47 | | G |
| 72 | 48 | | H |
| 73 | 49 | | I |
| 74 | 4A | | J |
| 75 | 4B | | K |
| 76 | 4C | | L |
| 77 | 4D | | M |
| 78 | 4E | | N |
| 79 | 4F | | O |
| 80 | 50 | | P |
| 81 | 51 | | Q |
| 82 | 52 | | R |
| 83 | 53 | | S |
| 84 | 54 | | T |
| 85 | 55 | | U |
| 86 | 56 | | V |
| 87 | 57 | | W |
| 88 | 58 | | X |
| 89 | 59 | | Y |
| 90 | 5A | | Z |
| 91 | 5B | | [ |
| 92 | 5C | | \ |
| 93 | 5D | | ] |
| 94 | 5E | | ^ |
| 95 | 5F | | _ |
| 96 | 60 | | ‘ |
| 97 | 61 | | a |
| 98 | 62 | | b |
| 99 | 63 | | c |
| 100 | 64 | | d |
| 101 | 65 | | e |
| 102 | 66 | | f |
| 103 | 67 | | g |
| 104 | 68 | | h |

**Tabella B.1** Codici ASCII per il PC (continua)

| Valore decimale | Valore esadecimale | Carattere di controllo | Carattere |
|---|---|---|---|
| 105 | 69 | | i |
| 106 | 6A | | j |
| 107 | 6B | | k |
| 108 | 6C | | l |
| 109 | 6D | | m |
| 110 | 6E | | n |
| 111 | 6F | | o |
| 112 | 70 | | p |
| 113 | 71 | | q |
| 114 | 72 | | r |
| 115 | 73 | | s |
| 116 | 74 | | t |
| 117 | 75 | | u |
| 118 | 76 | | v |
| 119 | 77 | | w |
| 120 | 78 | | x |
| 121 | 79 | | y |
| 122 | 7A | | z |
| 123 | 7B | | { |
| 124 | 7C | | ¦ |
| 125 | 7D | | } |
| 126 | 7E | | ~ |
| 127 | 7F | | ⌂ |
| 128 | 80 | | Ç |
| 129 | 81 | | ü |
| 130 | 82 | | é |
| 131 | 83 | | â |
| 132 | 84 | | ä |
| 133 | 85 | | à |
| 134 | 86 | | å |
| 135 | 87 | | ç |
| 136 | 88 | | ê |
| 137 | 89 | | ë |
| 138 | 8A | | è |
| 139 | 8B | | ï |

**Tabella B.1** Codici ASCII per il PC (continua)

| Valore decimale | Valore esadecimale | Carattere di controllo | Carattere |
|---|---|---|---|
| 140 | 8C | | î |
| 141 | 8D | | ì |
| 142 | 8E | | Ä |
| 143 | 8F | | Å |
| 144 | 90 | | É |
| 145 | 91 | | æ |
| 146 | 92 | | Æ |
| 147 | 93 | | ô |
| 148 | 94 | | ö |
| 149 | 95 | | ò |
| 150 | 96 | | û |
| 151 | 97 | | ù |
| 152 | 98 | | ÿ |
| 153 | 99 | | Ö |
| 154 | 9A | | Ü |
| 155 | 9B | | ¢ |
| 156 | 9C | | £ |
| 157 | 9D | | ¥ |
| 158 | 9E | | Pt |
| 159 | 9F | | ƒ |
| 160 | A0 | | á |
| 161 | A1 | | í |
| 162 | A2 | | ó |
| 163 | A3 | | ú |
| 164 | A4 | | ñ |
| 165 | A5 | | Ñ |
| 166 | A6 | | ª |
| 167 | A7 | | º |
| 168 | A8 | | ¿ |
| 169 | A9 | | ⌐ |
| 170 | AA | | ¬ |
| 171 | AB | | ½ |
| 172 | AC | | ¼ |
| 173 | AD | | ¡ |
| 174 | AE | | « |

**Tabella B.1** Codici ASCII per il PC (continua)

| Valore decimale | Valore esadecimale | Carattere di controllo | Carattere |
|---|---|---|---|
| 175 | AF | | » |
| 176 | B0 | | ░ |
| 177 | B1 | | ▒ |
| 178 | B2 | | ▓ |
| 179 | B3 | | │ |
| 180 | B4 | | ┤ |
| 181 | B5 | | ╡ |
| 182 | B6 | | ╢ |
| 183 | B7 | | ╖ |
| 184 | B8 | | ╕ |
| 185 | B9 | | ╣ |
| 186 | BA | | ║ |
| 187 | BB | | ╗ |
| 188 | BC | | ╝ |
| 189 | BD | | ╜ |
| 190 | BE | | ╛ |
| 191 | BF | | ┐ |
| 192 | C0 | | └ |
| 193 | C1 | | ┴ |
| 194 | C2 | | ┬ |
| 195 | C3 | | ├ |
| 196 | C4 | | ─ |
| 197 | C5 | | ┼ |
| 198 | C6 | | ╞ |
| 199 | C7 | | ╟ |
| 200 | C8 | | ╚ |
| 201 | C9 | | ╔ |
| 202 | CA | | ╩ |
| 203 | CB | | ╦ |
| 204 | CC | | ╠ |
| 205 | CD | | ═ |
| 206 | CE | | ╬ |
| 207 | CF | | ╧ |
| 208 | D0 | | ╨ |
| 209 | D1 | | ╤ |

**Tabella B.1** Codici ASCII per il PC (continua)

| **Valore decimale** | **Valore esadecimale** | **Carattere di controllo** | **Carattere** |
|---|---|---|---|
| 210 | D2 | | ╥ |
| 211 | D3 | | ╙ |
| 212 | D4 | | ╘ |
| 213 | D5 | | ╒ |
| 214 | D6 | | ╓ |
| 215 | D7 | | ╫ |
| 216 | D8 | | ╪ |
| 217 | D9 | | ┘ |
| 218 | DA | | ┌ |
| 219 | DB | | █ |
| 220 | DC | | ▄ |
| 221 | DD | | ▌ |
| 222 | DE | | ▐ |
| 223 | DF | | ▀ |
| 224 | E0 | | α |
| 225 | E1 | | β |
| 226 | E2 | | Γ |
| 227 | E3 | | π |
| 228 | E4 | | Σ |
| 229 | E5 | | σ |
| 230 | E6 | | μ |
| 231 | E7 | | τ |
| 232 | E8 | | Φ |
| 233 | E9 | | θ |
| 234 | EA | | Ω |
| 235 | EB | | δ |
| 236 | EC | | ∞ |
| 237 | ED | | Ø |
| 238 | EE | | ϵ |
| 239 | EF | | ∩ |
| 240 | F0 | | ≡ |
| 241 | F1 | | ± |
| 242 | F2 | | ≥ |
| 243 | F3 | | ≤ |
| 244 | F4 | | ⌠ |

**Tabella B.1** Codici ASCII per il PC (continua)

| **Valore decimale** | **Valore esadecimale** | **Carattere di controllo** | **Carattere** |
|---|---|---|---|
| 245 | F5 | | J |
| 246 | F6 | | ÷ |
| 247 | F7 | | ≈ |
| 248 | F8 | | ° |
| 249 | F9 | | • |
| 250 | FA | | · |
| 251 | FB | | √ |
| 252 | FC | | $^n$ |
| 253 | FD | | $^2$ |
| 254 | FE | | ■ |
| 255 | FF | | (spazio vuoto 'FF') |

**Tabella B.2** Codici secondari

| Secondo byte decimale | Secondo byte esadecimale | Significato |
|---|---|---|
| 3 | 03 | (Carattere nullo) NUL |
| 15 | 0F | SHIFT TAB |
| 16 | 10 | ALT Q |
| 17 | 11 | ALT W |
| 18 | 12 | ALT E |
| 19 | 13 | ALT R |
| 20 | 14 | ALT T |
| 21 | 15 | ALT Y |
| 22 | 16 | ALT U |
| 23 | 17 | ALT I |
| 24 | 18 | ALT O |
| 25 | 19 | ALT P |
| 30 | 1E | ALT A |
| 31 | 1F | ALT S |
| 32 | 20 | ALT D |
| 33 | 21 | ALT F |
| 34 | 22 | ALT G |
| 35 | 23 | ALT H |
| 36 | 24 | ALT J |
| 37 | 25 | ALT K |
| 38 | 26 | ALT L |
| 44 | 2C | ALT Z |
| 45 | 2D | ALT X |
| 46 | 2E | ALT C |
| 47 | 2F | ALT V |
| 48 | 30 | ALT B |
| 49 | 31 | ALT N |
| 50 | 32 | ALT M |
| 59 | 3B | Tasto funzione F1 |
| 60 | 3C | Tasto funzione F2 |
| 61 | 3D | Tasto funzione F3 |
| 62 | 3E | Tasto funzione F4 |
| 63 | 3F | Tasto funzione F5 |
| 64 | 40 | Tasto funzione F6 |
| 65 | 41 | Tasto funzione F7 |

**Tabella B.2** Codici secondari (continua)

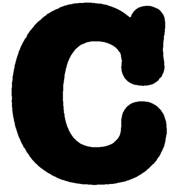
| Secondo byte decimale | Secondo byte esadecimale | Significato |
|---|---|---|
| 66 | 42 | Tasto funzione F8 |
| 67 | 43 | Tasto funzione F9 |
| 68 | 44 | Tasto funzione F10 |
| 71 | 47 | HOME |
| 72 | 48 | Cursore in alto (↑) |
| 73 | 49 | PG UP |
| 75 | 4B | Cursore a sinistra (←) |
| 77 | 4D | Cursore a destra (→) |
| 79 | 4F | END |
| 80 | 50 | Cursore in basso (↓) |
| 81 | 51 | PG DN |
| 82 | 52 | INS |
| 83 | 53 | DEL |
| 84 | 54 | F11 (SHIFT F1) |
| 85 | 55 | F12 (SHIFT F2) |
| 86 | 56 | F13 (SHIFT F3) |
| 87 | 57 | F14 (SHIFT F4) |
| 88 | 58 | F15 (SHIFT F5) |
| 89 | 59 | F16 (SHIFT F6) |
| 90 | 5A | F17 (SHIFT F7) |
| 91 | 5B | F18 (SHIFT F8) |
| 92 | 5C | F19 (SHIFT F9) |
| 93 | 5D | F20 (SHIFT F10) |
| 94 | 5E | F21 (CTRL F1) |
| 95 | 5F | F22 (CTRL F2) |
| 96 | 60 | F23 (CTRL F3) |
| 97 | 61 | F24 (CTRL F4) |
| 98 | 62 | F25 (CTRL F5) |
| 99 | 63 | F26 (CTRL F6) |
| 100 | 64 | F27 (CTRL F7) |
| 101 | 65 | F28 (CTRL F8) |
| 102 | 66 | F29 (CTRL F9) |
| 103 | 67 | F30 (CTRL F10) |
| 104 | 68 | F31 (ALT F1) |
| 105 | 69 | F32 (ALT F2) |

**Tabella B.2** Codici secondari (continua)

| Secondo byte decimale | Secondo byte esadecimale | Significato |
|---|---|---|
| 106 | 6A | F33 (ALT F3) |
| 107 | 6B | F34 (ALT F4) |
| 108 | 6C | F35 (ALT F5) |
| 109 | 6D | F36 (ALT F6) |
| 110 | 6E | F37 (ALT F7) |
| 111 | 6F | F38 (ALT F8) |
| 112 | 70 | F39 (ALT F9) |
| 113 | 71 | F40 (ALT F10) |
| 114 | 72 | CTRL PRTSC |
| 115 | 73 | CTRL cursore a sinistra (←) |
| 116 | 74 | CTRL cursore a destra |
| 117 | 75 | (→) |
| 118 | 76 | CTRL END |
| 119 | 77 | CTRL PG DN |
| 120 | 78 | CTRL HOME |
| 121 | 79 | ALT 2 |
| 122 | 7A | ALT 3 |
| 123 | 7B | ALT 4 |
| 124 | 7C | ALT 5 |
| 125 | 7D | ALT 6 |
| 126 | 7E | ALT 7 |
| 127 | 7F | ALT 8 |
| 128 | 80 | ALT 9 |
| 129 | 81 | ALT 0 |
| 130 | 82 | ALT – |
| 131 | 83 | ALT = |
| 132 | 84 | CTRL PG UP |

**Appendice**

# Messaggi d'errore BASIC



In questa appendice sono elencati tutti i messaggi d'errore del BASIC con i rispettivi codici numerici.

**1. NEXT without FOR**

Una variabile in un'istruzione NEXT non corrisponde a nessuna variabile delle istruzioni FOR precedentemente eseguite.

**2. Syntax error**

Una linea contiene qualche sequenza di caratteri non corretta (come parentesi non bilanciate, comandi e istruzioni con grafia errata, punteggiatura non esatta, e così via).

**3. RETURN without GOSUB**

È stata incontrata un'istruzione RETURN senza che fosse stata eseguita la relativa istruzione GOSUB.

**4. Out of data**

Un'istruzione READ ha tentato di leggere più dati di quanti se ne trovano nelle istruzioni DATA.

**5. Illegal function call**

Un parametro al di fuori dai limiti ammessi è stato passato ad una funzione. Oppure l'errore può essere il risultato di:

— Un parametro negativo o eccessivamente grande
— Una mantissa negativa con esponente non intero
— Una chiamata ad una funzione USR senza la definizione dell'indirizzo di partenza
— Un numero negativo di record in un'istruzione GET o PUT
— Un argomento improprio per un'istruzione o una funzione
— Un tentativo di intervenire su di un programma BASIC protetto
— Un tentativo di cancellare un numero di linea non esistente.

**6. Overflow**

Le dimensioni del dato eccedono lo spazio di memoria riservatogli dal formato indicato. Se l'errore di overflow si verifica con una variabile intera l'esecuzione termina, altrimenti viene utilizzato il massimo numero rappresentabile (detto infinito di sistema) con il segno appropriato e l'esecuzione continua. Se un numero è troppo piccolo per essere rappresentato dalla variabile intera, si verifica una condizione di *underflow*, ed in questo caso il risultato è 0 e l'esecuzione continua senza visualizzazione di messaggi d'errore.

**7. Out of memory**

Un programma è troppo lungo, ha troppi cicli FOR-NEXT o subroutine annidate, troppe variabili, espressioni troppo complicate, o istruzioni PRINT troppo complesse. L'istruzione CLEAR può essere usata all'inizio del programma per riservare più spazio per il programma.

**8. Undefined line number**

Un numero di linea a cui si fa riferimento in un'istruzione o in un comando non esiste nel programma.

**9. Subscript out of range**

Si fa riferimento ad un elemento di un array con un indice che eccede le dimensioni dell'array o con un errato numero di indici.

**10. Duplicate definition**

Avete cercato di definire due volte le dimensioni dello stesso array: questo può accadere in uno dei seguenti modi:

— Sono state utilizzate due istruzioni DIM per lo stesso array
— È stata usata un'istruzione DIM per un array che era già stato dimensionato a 10 per default

— È stata trovata un'istruzione OPTION BASE dopo che un array era stato dimensionato, sia con un'istruzione DIM che per default.

**11. Division by zero**

In un'espressione si cerca di dividere per zero o di elevare zero ad una potenza negativa. Viene utilizzato come risultato della divisione l'infinito di sistema con il segno del numeratore, oppure l'infinito di sistema con segno positivo come risultato dell'elevazione a potenza, e l'esecuzione continua.

**12. Illegal direct**

Un'istruzione non utilizzabile in modo diretto (per esempio, DEF FN) è stata usata ugualmente come comando in modo diretto.

**13. Type mismatch**

Avete passato un valore di tipo stringa dove ne era atteso uno numerico, o viceversa. Questo errore può venire inoltre causato dal tentativo di eseguire un'istruzione SWAP (scambio) tra variabili di tipo differente.

**14. Out of string space**

Il BASIC sta riservando spazio per stringhe fino a riempire tutta la memoria: questo messaggio significa che delle variabili di tipo stringa hanno portato il BASIC a superare lo spazio di memoria disponibile.

**15. String too long**

Avete cercato di creare una stringa più lunga dei 255 caratteri consentiti.

**16. String formula too complex**

Un'espressione di tipo stringa è troppo lunga o troppo complessa e deve essere spezzata in un'espressione più corta.

**17. Can't continue**

Avete tentato di usare CONT per continuare un programma che:

— È stato interrotto a causa di un errore
— È stato modificato durante un'interruzione dell'esecuzione
— Non esiste.

**18. Undefined user function**

Avete chiamato una funzione prima di averla definita con un'istruzione DEF FN.

**19. No RESUME**

Il programma è saltato ad una routine di intercettamento attiva come risultato di una condizione d'errore o di un'istruzione ERROR: questa routine non possiede un'istruzione RESUME oppure à stata trovata un'istruzione END, STOP, oppure RETURN prima dell'istruzione RESUME.

**20. RESUME without error**

È stata eseguita un'istruzione RESUME prima dell'inizio di una routine di intercettamento di un errore.

**22. Missing operand**

Un'espressione contiene un operatore, come + o AND, senza l'operando che deve seguirlo.

**23. Line buffer overflow**

Avete cercato di inviare al PC una linea con troppi caratteri.

**24. Device timeout**

Il BASIC non ha ricevuto informazioni da un dispositivo di I/O entro un predeterminato intervallo di tempo.

**25. Device fault**

Un'indicazione di un errore hardware è stata restituita da un'interfaccia: se questo avviene durante la trasmissione di dati ad un file di comunicazione, questo messaggio d'errore può indicare la mancanza di una o più linee di segnale.

**26. FOR without NEXT**

È stata eseguita un'istruzione FOR senza la corrispondente istruzione NEXT, ovvero un FOR era ancora attivo nel momento in cui è stata eseguita un'istruzione END, STOP o RETURN.

**27. Out of paper**

Manca la carta nella stampante, oppure questa non è accesa. Dovete inserire la carta, se necessario, verificare che la stampante sia opportunamente collegata e che, naturalmente, sia accesa.

**29. WHILE without WEND**

Ad un'istruzione WHILE non corrisponde alcuna istruzione WEND, ovvero è ancora attiva un'istruzione WHILE quando viene eseguita un'istruzione END, STOP o RETURN.

**30. WEND without WHILE**

È stata eseguita un'istruzione WEND prima della corrispondente WHILE.

**50. FIELD overflow**

Con un'istruzione FIELD avete tentato di definire più byte di quelli riservati per la lunghezza del record del file ad accesso diretto con un'istruzione OPEN, oppure è stata trovata la fine del buffer mentre era in esecuzione un'operazione di I/O sequenziale (PRINT#, WRITE#, INPUT# e così via) per un file ad accesso diretto.

**51. Internal error**

Si è verificato nel BASIC un malfunzionamento interno: spiegate al vostro rivenditore sotto quali condizioni è apparso il messaggio.

**52. Bad file number**

Un'istruzione fa riferimento ad un file non aperto (OPEN) o al di fuori dell'intervallo concesso per il numero di file specificati al momento dell'inizializzazione, oppure il nome del dispositivo nella specificazione del file è troppo lungo o non valido, oppure il nome stesso del file è troppo lungo o non valido.

**53. File not found**

Una delle seguenti istruzioni: LOAD, KILL, NAME, FILES, OPEN fa riferimento ad un file che non esiste sul disco inserito nel drive indicato.

**54. Bad file mode**

Avete cercato di usare un'istruzione PUT o GET con un file sequenziale o un file chiuso, oppure un'istruzione MERGE con un file non ASCII, oppure un'istruzione OPEN con un'opzione diversa da input, output, aggiunta, accesso diretto.

**55. File already OPEN**

Avete cercato di aprire per un output sequenziale o per un'aggiunta un file che era già aperto, oppure avete eseguito un'istruzione KILL con un file aperto.

**57. Device I/O error**

Si è verificato un errore durante un'operazione con un dispositivo di I/O, ed il DOS non è in grado di correggere l'errore e di proseguire.

**58. File already exists**

Il nome del file indicato in un'istruzione NAME è identico a quello di un file già presente sul dischetto.

**61. Disk full**

Tutto lo spazio di memoria del disco è utilizzato; quando si verifica quest'errore i file vengono chiusi.

**62. Input past end**

È stata eseguita un'istruzione INPUT per un file vuoto, o dopo che tutti i dati di un file sequenziale erano già stati letti. Per evitare quest'errore usate la funzione EOF per rivelare la fine del file. L'errore può anche verificarsi se cercate di leggere da un file che è stato aperto per l'output o per un'aggiunta.

**63. Bad record number**

In un'istruzione PUT o GET avete usato un numero di record maggiore del massimo consentito ($16777215 = 2^{24} - 1$) o uguale a zero.

**64. Bad file name**

È stata usata una forma non permessa per il nome del file in un'istruzione KILL, NAME, o FILES, per esempio un nome di file che inizia con un punto.

### 66. Direct statement in file

È stata trovata un'istruzione in modo diretto mentre si stava caricando o concatenando un file in formato ASCII: in questo caso l'operazione termina. Il file ASCII dovrebbe essere formato solo da istruzioni precedute da un numero di linea; l'errore può avvenire a causa di un carattere LF (line-feed) incontrato nell'input.

### 67. Too many files

Avete cercato di creare un nuovo file (usando SAVE o OPEN) quando il catalogo era pieno, o avete usato una specificazione di file non consentita.

### 68. Device unavailable

Avete cercato di aprire un file in un dispositivo che non esiste; cioè non avete il supporto hardware per il dispositivo (come un'interfaccia per stampante nel caso di una seconda o terza stampante), oppure avete disabilitato il dispositivo (per esempio avete usato l'opzione /C:0 nel comando di inizializzazione del BASIC per disabilitare i dispositivi di comunicazione).

### 69. Communication buffer overflow

È stata eseguita un'istruzione di input di dati quando il buffer di input era già pieno; quando incorrete in un errore di questo tipo dovete usare un'istruzione ON ERROR per riprendere l'input. Gli input successivi tenteranno di eliminare l'intoppo a meno che i caratteri non continuino ad essere ricevuti più in fretta di quanto il programma possa elaborarli. In questo caso, vi sono parecchie possibilità:

— Aumentare le dimensioni del buffer di comunicazione quando avviate il BASIC con l'opzione /C.
— Implementare un protocollo di tipo hand-shake con l'altro computer per impedirgli di inviare altri caratteri finché il PC non possa accettarli
— Usare un baud rate inferiore
— Riscrivere la parte del programma che provoca il rallentamento per renderla più efficiente.

### 70. Disk write-protect

Avete tentato di scrivere su di un dischetto protetto contro la sovrascrittura.

**71. Disk not ready**

Lo sportello del drive non è chiuso, oppure il disco non è inserito. Inserite nel drive il dischetto corretto e proseguite il programma.

**72. Disk media error**

Il controller o la scheda di interfaccia ha trovato un difetto nell'hardware o nel collegamento; di solito ciò significa che il dischetto non è in ordine. Copiate tutti i file in un nuovo dischetto e riformattate il dischetto in questione; se la formattazione non riesce, il dischetto deve essere scartato.

**73. Advanced feature**

Il vostro programma usa una configurazione dell'Advanced BASIC mentre voi state usando il Disk BASIC. Caricate l'Advanced BASIC e riavviate il programma.

**74. RENAME across disks**

Avete cercato di cambiare nome al file, ma avete specificato il disco sbagliato. L'operazione non viene eseguita.

**75. Path/file access error**

Durante un'operazione OPEN, NAME, MKDIR, CHDIR, o RMDIR, si è tentato di usare un cammino o un nome di file per un file non accessibile. Per esempio, potete aver tentato di aprire un directory oppure di aprire un file a sola lettura per delle operazioni di scrittura, o di rimuovere il directory corrente. L'operazione non viene completata.

**76. Path not found**

Durante un'operazione OPEN, MKDIR, CHDIR, o RMDIR, il DOS non è in grado di trovare un cammino specificato. L'operazione non viene completata.

**– Incorrect DOS version**

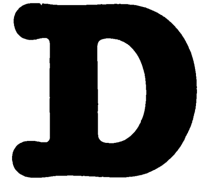
Il comando che voi avete inviato richiede una versione del DOS diversa da quella che avete caricato.

**– Unprintable error**

Non è disponibile nessun messaggio d'errore per la condizione che si è verificata. Solitamente è causata da un'istruzione ERROR con un codice d'errore indefinito.

Appendice

# Sommario dei comandi DOS



Il seguente sommario contiene i comandi del DOS 2.0 e 2.1; esistono quattro tipi fondamentali di comandi DOS: quelli standard, chiamati semplicemente *comandi*; *comandi batch*, usati all'interno di file batch; *comandi avanzati*, che sono di notevole interesse per gli utenti esperti del DOS; infine *utility*, che forniscono all'utente speciali funzioni. I comandi sono elencati in ordine alfabetico e la categoria a cui appartengono è scritta al loro fianco.

## **ASSIGN** Comando

*Sintassi:*

ASSIGN [*d1*=*d2*...]

Scambia temporaneamente le lettere che identificano i drive. Ogni successivo accesso al drive *d1* userà invece il drive *d2*. Se non vengono inclusi parametri, tutti gli altri eventuali riassegnamenti di drive vengono riportati alla condizione normale.

## **BACKUP** Comando

*Sintassi:*

BACKUP [*specfile*] *d:* [/S] [/M] [/A] [/D:*mm-gg-aa*]

Consente di effettuare la copia di sicurezza di uno o più file, dal disco rigido ad un dischetto. Il parametro *specfile*, opzionale, indica il file di cui

si vuole fare la copia di backup. Possono essere usati i caratteri jolly ? e *. Se non viene specificato il nome del file, il DOS procederà a copiare tutti i file del directory.
Il drive destinazione viene specificato nell'opzione *d:*; l'opzione /S (subdirectory) permette di copiare tutti i file del subdirectory oltre a quelli del directory specificato. L'opzione /M farà copiare solo quei file che sono stati modificati dopo l'ultimo backup. /A aggiunge i file di backup nel dischetto specificato che già contiene altri file di backup. Infine /D:*mm-gg-aa* (data) vi consente di fare la copia di backup solo per quei file che sono stati modificati in quella data o successivamente.

## **BREAK** Comando

*Sintassi:*
BREAK [ON|OFF]

Durante l'esecuzione di un programma, il comando BREAK ON produce un controllo per il tasto CTRL BREAK ad ogni richiesta DOS da parte del programma stesso. Questo consentirà di interrompere l'esecuzione di un programma più velocemente del normale. Comunque, il risultato è un'esecuzione leggermente più lenta di tutti i programmi. BREAK OFF, invece, disabilita questo comando ed ordina al DOS di controllare il tasto CTRL BREAK solo durante operazioni di accesso a schermo, tastiera, stampante o ACA. Infine, BREAK senza alcun parametro visualizza lo stato corrente del controllo del tasto CTRL BREAK.

## **CHDIR** Comando

*Sintassi:*
CHDIR [[*d:*] *cammino*]
CD [[*d:*] *cammino*]

Questi comandi cambiano il directory corrente del drive specificato, o di default, nel directory indicato in *cammino*. Il directory radice è specificato con il simbolo " \ " (backslash), il directory padre viene indicato con il simbolo "..". Il comando CD è un'abbreviazione di CHDIR. Se il parametro *cammino* non viene specificato, viene visualizzato il cammino del directory corrente.

## **CHKDSK** Comando

*Sintassi:*
CHKDSK [*d:*] [*specfile*] [/F] [/V]

Effettua una verifica del directory e dei file del drive *d:* e produce un rapporto sullo stato del disco *d:* e della memoria. Se *d:* viene omesso, la verifica e il rapporto sullo stato del disco vengono eseguiti sul drive di default.
Con l'argomento *specfile* visualizza il numero di aree non contigue occupate dal file specificato. Nel parametro *specfile* sono permessi i caratteri jolly ? e *.
L'opzione /F consente al comando di correggere gli errori trovati nel directory della tabella di allocazione dei file; questo per permettere di liberare spazio sul disco. L'opzione /V visualizza i nomi dei file esaminati.

## **CLS** Comando

*Sintassi:*
CLS

Cancella lo schermo.

## **COMP** Comando

*Sintassi:*
COMP [*specfile1*] [*specfile2*]

Confronta il contenuto di *specfile1* con quello di *specfile2*. Se le specificazioni vengono omesse, COMP vi richiede i nomi dei due file da confrontare. In entrambi i nomi di file è consentito l'uso dei caratteri jolly ? e *.

## **COPY** Comando

*Sintassi:*
COPY *specfile1* [*specfile2*]

Copia il file indicato in *specfile1* nel file indicato in *specfile2*. Se il drive destinazione non viene indicato, il comando farà riferimento al drive di default; se invece non vengono specificati il nome del file e la sua estensione, il nuovo file avrà lo stesso nome e la stessa estensione del file sor-

gente. Sorgente e destinazione possono essere specificate come dispositivi riservati.

*Sintassi:*
COPY *specfile1* [*specfile2*] [/V]

L'opzione /V copia *specfile1* in *specfile2* e verifica che il file sia stato copiato correttamente.

*Sintassi:*
COPY *specfile1*+*specfile2*+...+*specfilex* [*specfiley*]

Copia e concatena i file da *specfile1* a *specfilex* nel file destinazione; se questo non viene indicato, verrà usato nuovamente *specfile1*.

*Sintassi:*
COPY [/A] [/B] *specfile1* [/A] [/B] [*specfile2*]

L'opzione /A comanda a COPY di trattare il file sorgente e/o il file destinazione come file ASCII o file testo. Il file sorgente con l'opzione /A verrà copiato fino a quando non si incontri il carattere ASCII EOF (end-of-file). (il carattere EOF è CTRL Z, in esadecimale 1A). Il comando COPY aggiungerà il carattere EOF alla fine del file destinazione se è stato così specificato dall'opzione /A.
Il parametro /B specificato per un file sorgente farà copiare l'intero file. Invece, per un file destinazione, l'opzione /B indica che non deve essere aggiunto alcun carattere EOF.
Il valore di default è /B quando non viene specificata la concatenazione di file, mentre è /A se la concatenazione deve essere eseguita.

## **CTTY** Comando avanzato

*Sintassi:*
CTTY *dispositivo*

Vi consente di usare con il PC un terminale esterno. Il parametro *dispositivo* può essere AUX, COM1, o COM2 per sostituire l'input e l'output della tastiera e dello schermo con quelli di un terminale esterno. L'uso del *dispositivo* CON riporta l'input e l'output standard alla tastiera e allo schermo del PC.

## **DATE** Comando

*Sintassi:*
DATE [*mm-gg-aa*]
DATE [*mm/gg/aa*]

Fornisce al sistema la data specificata. Se non viene inserito alcun valore, viene visualizzata la data corrente e vi viene richiesto di inserire una nuova data. Per lasciare la data così com'è premete ENTER in risposta al prompt.

## **DEBUG** Utility

*Sintassi:*
DEBUG [*specfile*]

Richiama il programma DEBUG e opzionalmente carica il file indicato in *specfile*.

## **DEL** Comando

*Sintassi:*
DEL [*specfile*]

Questo comando è funzionalmente identico a ERASE: cancella il file indicato in *specfile* dal directory e dal drive correnti o definiti. Se non viene passato il nome del file in *specfile*, il DOS assume come nome di file *.*. Lo specificatore di file *.* fa sì che tutti i file negli appropriati disco e directory vengano cancellati. Ogni tentativo di usare lo specificatore *.* provocherà una richiesta di conferma dell'azione da parte del DOS, prima di cancellare effettivamente i file.

## **DIR** Comando

*Sintassi:*
DIR [*specfile*] [/P] [/W]

Visualizza un catalogo elencando nomi, lunghezza e data di creazione dei file e dei subdirectory contenuti nel drive e nel catalogo specificati (o nel drive di default e nel catalogo corrente se non specificati). Se vengono indicati il nome del file o l'estensione, la lista conterrà solo quei file che

hanno informazioni uguali a quelle date in *specfile*. È consentito l'uso dei caratteri jolly ? e *. I subdirectory saranno contrassegnati con l'identificatore <DIR> nel campo della dimensione del file.
L'opzione /P sospende la visualizzazione delle informazioni quando lo schermo è pieno. Per continuare con l'elenco del catalogo dopo la sospensione basta premere un tasto qualsiasi.
L'opzione /W produce l'elenco condensato di un catalogo con solo i nomi dei file. Ogni riga dell'elenco conterrà cinque nomi. L'opzione /W è raccomandata solo per sistemi con schermo a 80 colonne.

## **DISKCOMP** Comando

*Sintassi:*
DISKCOMP [*d1:*] [*d2:*] [/1] [/8]

Confronta il dischetto nel primo drive specificato con il dischetto nel secondo drive. Se il drive non viene indicato, il comando farà riferimento al drive di default. Questa operazione garantisce che i due dischetti siano identici.
L'opzione /1 limita il confronto solo alla prima faccia del dischetto, anche se questo è un dischetto a doppia faccia. Il parametro /8 confronterà solo 8 settori per traccia, invece dei 9 settori per traccia di default.

## **DISKCOPY** Comando

*Sintassi:*
DISKCOPY [*d1:*] [*d2:*] [/1]

Copia l'intero contenuto del dischetto nel primo drive specificato nel dischetto contenuto nel secondo drive. Se non viene indicato alcun drive, si utilizzerà il drive di default.
L'opzione /1 limita l'operazione di copia del disco alla prima faccia del dischetto anche se il disco sorgente è a doppia faccia.

## **ECHO** Comando batch

*Sintassi:*
ECHO [ON|OFF|*messaggio*]

Il comando ECHO OFF arresta la visualizzazione sullo schermo dei comandi (ma non dei messaggi DOS) nel momento in cui vengono eseguiti

dal file batch. ECHO ON riabilita l'eco dei comandi. L'opzione *messaggio* vi consente di visualizzare sullo schermo uno specifico testo senza tener conto del fatto che sia attivo ECHO ON o ECHO OFF. Se nessun parametro è assegnato a ECHO, viene visualizzato il corrente stato di ECHO.

## **EDLIN** Utility

*Sintassi:*
EDLIN *specfile*

Richiama il programma EDLIN per creare o modificare il file indicato in *specfile.*

## **ERASE** Comando

*Sintassi:*
ERASE [*specfile*]

Cancella il file indicato in *specfile* dal drive e dal catalogo definito o di default. Se non viene passato il nome del file in *specfile,* il DOS assume come nome di file *.*. Lo specificatore di file *.* fa sì che tutti i file nel disco e catalogo vengano cancellati. Ogni tentativo di usare lo specificatore *.* provocherà una richiesta di conferma dell'azione da parte del DOS, prima di cancellare i file.

## **EXE2BIN** Comando avanzato

*Sintassi:*
EXE2BIN *specfile1* [*specfile2*]

Converte il file eseguibile (.EXE) indicato in *specfile1* in un'immagine assoluta di memoria (.COM) e carica quest'ultima nel file di output indicato in *specfile2.*

## **File batch**

*Sintassi:*
[*d:*] *nomefile* [*parametri*]

Avvia l'esecuzione del file batch chiamato *nomefile* e provoca l'esecuzione dei comandi in esso contenuto. Parametri fittizi specificati come %*n*,

con *n* da 0 a 9, all'interno del file batch vengono rimpiazzati dagli specificati parametri effettivi nell'ordine in cui appaiono. Il parametro fittizio %0 è sempre rimpiazzato da [*d:*]*nomefile*.
Se il disco del DOS contiene un file AUTOEXEC.BAT nel directory radice, i comandi per file batch all'interno del file stesso verranno eseguiti automaticamente ad ogni accensione o reinizializzazione del sistema.

## **FIND** Comando

*Sintassi:*
FIND [/V] [/C] [/N] "*stringa*" [*specfile...*]

Ricerca all'interno dei file indicati in *specfile* tutte le volte in cui compare il parametro *stringa*. Ogni linea che contenga questa stringa viene inviata al dispositivo standard di output. Se *specfile* viene omesso, l'input viene atteso da un pipe del DOS.
L'opzione /V causa l'invio al dispositivo standard di output di tutte le linee in input che non contengono *stringa*. L'opzione /C, invece, mostra il numero di volte in cui *stringa* è presente, invece che le linee di testo contenenti *stringa*. Infine, l'opzione /N invia in output il numero e ogni linea in cui viene trovata *stringa*.

## **FOR** Comando batch

*Sintassi:*
FOR %%*variabile* IN (*elenco*) DO *comando*

Consente una ripetizione di comandi per ogni dato dell'*elenco*. In ogni iterazione %%*variabile* viene posto uguale al successivo dato di (*elenco*) e viene quindi eseguito *comando*. *comando* può usare il valore di %%*variabile* come un parametro variabile.

## **FORMAT** Comando

*Sintassi:*
FORMAT [*d:*] [/S] [/1] [/8] [/V] [/B]

Inizializza il dischetto nel drive specificato. Se non viene specificato il drive viene utilizzato quello di default. FORMAT richiede una conferma per dare inizio al processo. Dovete formattare tutti i dischetti nuovi ed il disco rigido prima che questi possano venir usati dal DOS. Il comando

DISKCOPY formatta il dischetto durante il processo di copia se non lo avete già fatto con il comando FORMAT.
L'opzione /S provoca la copia dei file del sistema operativo (IBMBIO. COM, IBMDOS.COM e COMMAND.COM) nel dischetto da formattare.
L'opzione /1 limita la formattazione ad una singola faccia del dischetto, mentre l'opzione /8 formatta il disco con 8 settori per traccia. I valori di default sono: formattazione a doppia faccia e 9 settori per traccia).
L'opzione /V vi consente di fornire al disco da formattare una volume label, cioè un nome per il dischetto. Questa label può essere visualizzata con i comandi DIR e VOL.
Il parametro /B formatta il dischetto con 8 settori per traccia, riservando spazio per i file di sistema IBMBIO.COM e IBMDOS.COM. I file veri e propri possono essere inseriti nel dischetto in seguito per mezzo del comando SYS.

## **GOTO** Comando batch

*Sintassi:*
GOTO *label*

Vi permette di trasferire il flusso di esecuzione di un file batch in un qualsiasi punto. Il parametro *label* ordina al DOS di cercare nel file batch fino al ritrovamento della *label* corrispondente. L'esecuzione del file batch continua quindi con il comando immediatamente seguente la linea identificata da *label*.

## **GRAPHICS** Comando

*Sintassi:*
GRAPHICS

Predispone il DOS alla stampa della pagina grafica sulla stampante a matrice IBM o su una stampante Epson MX. La stampa potrà essere ottenuta premendo i tasti SHIFT PRTSC.

## **IF** Comando batch

*Sintassi:*
IF [NOT] *condizione comando*

Consente l'esecuzione condizionata di comandi in un file batch. Se la *condizione* viene valutata come vera, il *comando* viene eseguito, altrimenti

viene saltato. Se è inclusa anche l'opzione NOT, il *comando* viene eseguito solo se la *condizione* viene valutata come falsa.
Il parametro *condizione* può essere ERRORLEVEL *numero*, *stringa1*= = *stringa2*, oppure EXIST *specfile*; ERRORLEVEL è vero se il precedente programma aveva un codice di uscita maggiore o uguale a *numero*. (Alcuni comandi e programmi DOS una volta interrotti o terminata l'esecuzione restituiscono un valore al DOS; questo valore viene detto codice d'uscita). *stringa1*= =*stringa2* risulterà vero se *stringa1* è identica a *stringa2*. EXIST *specfile* è vera se *specfile* è presente nel drive indicato.

## **LINK** Utility

*Sintassi:*
LINK

Richiama il programma LINK, che combina più moduli oggetto in programmi eseguibili.

## **MKDIR** Comando

*Sintassi:*
MKDIR [*d*:] [*cammino* \ ]*nomedir*
MD [*d*:] [*cammino* \ ]*nomedir*

Crea un nuovo subdirectory chiamato *nomedir* nel disco specificato (o nel disco di default se non specificato). Il nuovo subdirectory viene posto nel directory indicato in *cammino*. Un carattere \ davanti a *cammino* indica la radice della struttura ad albero. I comandi MKDIR e MD sono identici.

## **MODE** Comando

*Sintassi:*
MODE [*n*], *m*[,T]

Il parametro opzionale *n*, che può essere uguale a 40, 80, BW40, BW80, CO40, CO80, oppure MONO, assegna al video le dimensioni di 40 o 80 caratteri per riga. BW e CO predispongono l'adattatore del video rispettivamente alla scheda Colore/Grafica in bianco e nero o a colori, rispettivamente. L'opzione MONO, invece, predispone il video all'interfaccia per video monocromatico.

Il parametro $m$ fa scorrere la videata a destra ($m$=R) o a sinistra ($m$=L) per un corretto allineamento. Agisce solo sulla scheda Colore/Grafica. Il parametro opzionale T genera una videata di testo per la verifica dell'allineamento.

*Sintassi:*
MODE LPT#: [*n*] [,[*m*] [,P]]

Inizializza la stampante specificata (# =1, 2 o 3) per operazioni con $n$ caratteri per linea ($n$=80 o 132) e $m$ linee per pollice ($m$=6 o 8). Il valore di default all'accensione è $n$=80 e $m$=6. L'opzione P specifica la ripetizione ciclica dell'invio dei dati alla stampante in caso di errore di timeout.

*Sintassi:*
MODE COM*n*:*baud* [,*parità* [,*bit dati* [,*bit stop* [,P]]]]

Predispone l'ACA specificata da $n$ ($n$=1 o 2) per operazioni alla velocità di trasmissione indicata in *baud* (*baud*=110, 150, 300, 600, 1200, 2400, 4800, o 9600). I parametri di protocollo vengono predisposti in accordo con i valori di *parità* (N=nessuna parità, O=parità dispari, E=parità pari), con *bit dati* (7 o 8), e *bit stop* (1 o 2). I valori di default sono parità pari, 7 *bit dati* e 1 *bit stop* (due *bit stop* a 110 baud). L'opzione P, che indica che l'ACA dev'essere usata con una stampante seriale, provoca la continua ripetizione in caso di errore di timeout.

*Sintassi:*
MODE LPT#: = COM*n*

Reindirizza l'output della stampante parallela inviato alla stampante specificata con # (# =1, 2 o 3) all'interfaccia asincrona indicata da $n$ ($n$=1 o 2).

## **MORE** Comando

*Sintassi:*
MORE<*specfile*

Questo comando permette di leggere dati da *specfile* e li invia al dispositivo standard di output, facendo una pausa dopo ogni pagina; il messaggio MORE viene inviato al dispositivo di output che rimane in attesa che un qualsiasi tasto della tastiera venga premuto prima di riprendere.

## **PATH** Comando

*Sintassi:*
PATH [*d:*] *cammino* [[;[*d:*] *cammino*]...]

Indica i directory attraverso i quali devono essere ricercati comandi o file batch ogni volta che non si trovino nel directory corrente. Potete indicare una lista di drive e di cammini: se si omettono tutti i cammini, PATH visualizza il cammino corrente.

## **PAUSE** Comando batch

*Sintassi:*
PAUSE [*messaggio*]

Sospende l'esecuzione di un file batch e visualizza il testo contenuto in *messaggio*, se indicato. L'esecuzione del file batch può essere ripresa premendo un qualsiasi tasto, purché diverso da CTRL BREAK che, invece, conclude definitivamente l'esecuzione del file batch.

## **Pipe**

*Sintassi:*
*comandoA*|*comandoB*

L'uso di pipe (che in inglese significa tubo, conduttura) vi permette di "concatenare" diversi programmi. L'output standard (normalmente inviato allo schermo) di *comandoA* viene inviato come input standard (normalmente ricevuto da tastiera) di *comandoB*.

## **PRINT** Comando

*Sintassi:*
PRINT [*specfile* [/T] [/C] [/P]...]

Crea una coda di stampa di non più di dieci file di testo permettendovi di continuare a compiere altre operazioni sul computer. Se non vengono specificati parametri, verranno visualizzati i nomi dei file attualmente presenti nella coda di stampa. La prima volta che eseguite PRINT il DOS vi richiederà il nome del dispositivo di output (PRN di default); dispositivi di output utilizzabili sono LPT1:, LPT2:, LPT3:, PRN:, COM1:, COM2:, AUX:, ecc.

L'opzione /T rimuove tutti i file dalla coda di stampa. L'opzione /C rimuove i file specificati dalla coda di stampa. Tutti i file, da quello immediatamente precedente /C alla fine della linea di comando o al successivo /P verranno cancellati dalla coda. L'opzione /P permette di aggiungere alla coda di stampa i file specificati dopo l'opzione /C, sulla stessa linea. Tutti i file, da quello precedente /P fino alla fine della linea di comando o alla successiva opzione /C verranno aggiunti alla coda di stampa.

## **PROMPT** Comando avanzato

*Sintassi:*
PROMPT [*stringa*]

Vi consente di assegnare un nuovo prompt di sistema utilizzato dal DOS. Il parametro *stringa* consiste di un qualsiasi numero di stringhe che possono essere combinate per formare il prompt desiderato. Omettendo questo parametro verrà riassegnato il normale prompt del DOS.
I seguenti caratteri, quando siano preceduti dal carattere $, hanno un significato speciale all'interno di *stringa*:

b carattere "¦"
d data corrente
e carattere ESCAPE
g carattere ">"
h sequenza di caratteri BACKSPACE e DELETE
l carattere "<"
n specificatore del drive di default
p directory corrente del drive di default
q carattere "="
t ora di sistema
v il numero di versione del DOS
_ a capo (CR/LF).

## **RECOVER** Comando

*Sintassi:*
RECOVER *specfile*
RECOVER *d:*

Recupera il file indicato in *specfile* da un disco che ha un settore difettoso. Del file che contiene il settore difettoso potete ricuperare tutto tranne il contenuto della parte di disco rovinata. Se il settore difettoso contiene

il directory, tutti i file sul disco possono essere recuperati specificando solo *d:*, invece di *specfile*. Usate RECOVER *d:* con grande attenzione, perché rende difficilmente identificabili i file del disco in *d:*.

## Redirezione dell'I/O

*comando*>*specfile*
*comando*>>*specfile*
*comando*<*specfile*

L'input standard dalla tastiera e l'output standard sullo schermo possono essere rediretti durante l'esecuzione di un programma o di un comando DOS. L'assegnamento > invia l'output di *comando* al file indicato in *specfile*, invece che allo schermo. Se *specfile* esisteva già prima di questa operazione andrà perso il suo contenuto precedente.
L'assegnamento >> redirige l'output di *comando*, ma aggiunge tutti gli output alla fine del contenuto di *specfile*, se questa esiste. Il precedente contenuto di *specfile* non viene perso. Infine, l'assegnamento < fa sì che l'input standard di *comando* venga letto da *specfile* invece che dalla tastiera.

## **REM** Comando batch

*Sintassi:*
REM [*messaggio*]

Visualizza commenti per l'operatore durante l'esecuzione di file batch.

## **RENAME** Comando

*Sintassi:*
RENAME *specfile nomefile* [*.est*]

Cambia il nome del file indicato in *specfile* in *nomefile*[*.est*]. È consentito l'uso dei caratteri jolly ? e *. Può essere utilizzata, in luogo di RENAME, la sua abbreviazione REN.

## **RESTORE** Comando

*Sintassi:*
RESTORE *d:* [*d:*] [*cammino*] [*nomefile*] [*.est*] [/S] [/P]

Ricopia uno o più file da dischetto a disco rigido. I file da ricopiare devono essere stati caricati sul dischetto col comando BACKUP. Il primo parametro specifica il drive del dischetto di backup. Il secondo parametro è il file su disco rigido che volete ripristinare. È consentito l'uso dei caratteri jolly ? e *.
L'opzione /S causa la sostituzione di tutti i file di backup in ogni subdirectory in aggiunta ai file del directory specificato. L'opzione /P vi richiederà la conferma dell'operazione RESTORE per ognuno dei file prima di eseguirla.

## **RMDIR** Comando

*Sintassi:*
RMDIR [*d:*] *cammino*
RD [*d:*] *cammino*

Rimuove un subdirectory dal disco specificato (o dal disco di default se non indicato). Prima di essere cancellato, il directory dev'essere stato svuotato da tutti i file e i subdirectory (eccetto i termini con "." e ".."). Il comando abbreviato RD può essere usato al posto di RMDIR.

## **SET** Comando avanzato

*Sintassi:*
SET [*nomevar*=[*stringa*]]

Le stringhe contenenti le variabili d'ambiente definiscono lo stato di alcuni parametri fondamentali per il DOS o i programmi applicativi, come ad esempio PATH, COMSPEC, PROMPT, BREAK.
Lo stato corrente può essere visualizzato da SET senza alcun parametro. SET *nomevar*= cancella la definizione corrente di *nomevar*, mentre la sintassi completa definisce una nuova variabile.

## **SHIFT** Comando batch

*Sintassi:*
SHIFT

Consente ad un file batch di accedere a più di dieci parametri. I parametri vengono passati al file batch nella linea dei comandi batch. Ogni comando SHIFT provoca la sostituzione del corrente parametro %0 con il parametro %1 e così via, mentre il parametro %9 nella linea dei comandi batch viene rimpiazzato dall'undicesimo parametro e poi dal dodicesimo e così via.

## **SORT** Comando

*Sintassi:*
SORT [/R] [/+*n*] [<*specfile*]

Legge dati dall'input standard, li ordina secondo il codice ASCII e quindi scrive i dati ordinati sull'output standard. L'ordinamento è fatto su linee di testo. L'opzione /R produce l'ordinamento inverso. L'opzione /+*n* vi consente di effettuare l'ordinamento a partire dall'*n*-esimo carattere di ogni linea. Il valore di default è $n=1$.

## **SYS** Comando

*Sintassi:*
SYS *d:*

Copia i file del sistema operativo IBMBIO.COM e IBMDOS.COM dal drive di default al drive specificato in *d:*. Il disco destinazione deve essere stato formattato con i comandi FORMAT *d:*/S o FORMAT *d:*/B.

## **TIME** Comando

*Sintassi:*
TIME [*hh*[:*mm*[:*ss*[:*xx*]]]]

Assegna l'ora specificata come ora corrente. L'ora viene tenuta aggiornata finché il sistema non viene spento o reinizializzato. L'ora corrente viene usata per marcare la creazione o la modifica di file e directory; può

essere esaminata omettendo i parametri in TIME e premendo poi ENTER in risposta alla richiesta dell'ora corrente.

## **TREE** Comando

*Sintassi:*
TREE [*d:*] [/F]

Visualizza la struttura del directory del disco contenuto nel drive specificato (o nel drive di default se non specificato). L'opzione /F elencherà inoltre i nomi di tutti i file di ogni subdirectory. Questo comando vi consente di esaminare la struttura ad albero del disco.

## **TYPE** Comando

*Sintassi:*
TYPE *specfile*

Visualizza il contenuto del file indicato in *specfile*. Normalmente viene utilizzato per visualizzare il contenuto dei file di testo.

## **VER** Comando

*Sintassi:*
VER

Visualizza il numero della versione del DOS utilizzata.

## **VERIFY** Comando

*Sintassi:*
VERIFY [ON | OFF]

VERIFY ON ordina al DOS di verificare automaticamente ogni operazione di scrittura su disco. Ciò assicura la correttezza di ogni scrittura, ma rallenta l'attività del sistema. Il comando VERIFY OFF, invece, disabilita questa verifica automatica. VERIFY senza alcun parametro visualizza lo stato corrente di VERIFY.

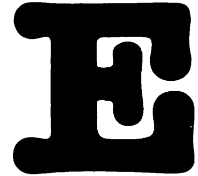
## **VOL** Comando

*Sintassi:*
VOL [*d:*]

Visualizza la volume label del disco in *d:* (cioè il nome del disco assegnato da FORMAT). Se il drive non viene specificato, viene visualizzata la volume label del disco nel drive di default.

Appendice

# Messaggi DOS



Questa appendice comprende un elenco di comandi DOS seguiti dai più comuni messaggi che possono apparire durante la loro esecuzione.

## BACKUP

*** * * Backing up files to diskette XX * * ***
(Sto copiando i file nel disco *XX*)

Questo messaggio sarà seguito da un elenco dei file che vengono copiati sul disco *XX*.

**Diskette is not a backup diskette**
(Il disco non è un disco di backup)

Questo dischetto non è stato creato dal comando BACKUP.

**Insert backup diskette XX in drive Y:**
**Warning! Diskette files will be erased**
**Strike any key when ready**
(Inserire il disco di backup *XX* nel drive *Y:*
Attenzione! I file presenti sul disco verranno cancellati
Premere un qualsiasi tasto per iniziare)

Inserire il disco successivo nella sequenza del backup. I dischetti devono essere stati formattati dal DOS; premere un tasto qualsiasi per continuare il backup.

**Warning! No files were found to back up**
(Attenzione! non sono stati trovati file da copiare)

Non ci sono file sul disco rigido che corrispondono alla specificazione indicata per il BACKUP.

## Batch

**FOR cannot be nested**
(Il comando FOR non può essere annidato)

Non potete indicare comandi multipli FOR su un'unica linea di comandi BATCH.

**Label not found**
(Label non trovata)

Un comando GOTO richiama una label che non compare nel file batch.

## CHKDSK

**All specified file(s) are contiguous**
(Tutti i file indicati sono contigui)

I (o il) file che avete indicato sono scritti su disco in modo consecutivo; non c'è cioè frammentazione dei file.

**Allocation error for file, size adjusted**
(Errore di allocazione del file, dimensione ridotta)

È stato trovato un numero di settore non valido nella tabella di allocazione del file indicato; il file è stato troncato al termine dell'ultimo settore valido.

**Contains invalid cluster, file truncated**
(Contiene un cluster non valido; il file è stato troncato)

Il file indicato contiene un puntatore all'area dei dati non valido. Il file è stato troncato al termine dell'ultimo blocco valido di dati. Il troncamento si verifica solo se è stata specificata l'opzione /F.

**Contains XXX noncontiguous blocks**
(Contiene *XXX* blocchi non contigui)

Il file indicato non è scritto sul disco in modo consecutivo, ma contiene *XXX* "frammenti". Questa non è una condizione d'errore.

**Convert directory to file (Y/N)?**
(Devo convertire il directory in un file (Y/N)?)

Il directory indicato non ha un formato valido. Rispondendo affermativamente alla richiesta, il directory viene trasformato in un file che può essere esaminato e modificato per mezzo di EDLIN.

**Convert lost chains to files (Y/N)?**
(Devo convertire in un file le catene perdute (Y/N)?)

Rispondendo affermativamente alla domanda, si ordina a CHKDSK di convertire in file i dati che si trovano nei blocchi "perduti"; rispondendo no, invece, si liberano questi blocchi per l'allocazione di nuovi file. In realtà si verificano cambiamenti in questi blocchi solo quando CHKDSK è stato richiamato con l'opzione /F.

**Disk error writing FAT X**
(Errore di scrittura FAT *X*)

Si è verificato un errore mentre CHKDSK stava scrivendo la tabella di allocazione dei file (*File Allocation Table*, cioè FAT). *X* indica in quale delle due copie della tabella si è verificato l'errore; se $X=1$ e 2, entrambe le tabelle sono errate ed il disco è inutilizzabile.

**Entry has a bad attribute (or size or link)**
(Il termine ha attributo, dimensione oppure collegamento sbagliati)

Uno dei subdirectory "." o ".." è in errore. CHKDSK cercherà di correggere l'errore se è stata indicata l'opzione /F.

**Error found, F parameter not specified**
**Corrections will not be written to disk**
(È stato trovato un errore, il parametro F non è stato indicato. Le correzioni non vengono scritte sul disco)

Nessuna delle correzioni indicate da CHKDSK viene applicata non essendo stato specificato il parametro/F.

**File is cross-linked: on cluster XX**
(File incrociati nel cluster *XX*)

Lo stesso cluster (cioè un gruppo di settori contigui sul disco) è stato utilizzato per più di un file. Dovete eseguire una correzione come segue:

1. Utilizzare COPY per fare copie di entrambi i file
2. Cancellare con ERASE i file originali
3. Controllare la correttezza dei file.

**First cluster number is invalid, entry truncated**
(Il numero del cluster non è valido, il file è stato troncato)

Il file indicato contiene un puntatore all'area dei dati non valido. Il file è stato troncato ad una lunghezza zero se era stata specificata l'opzione /F.

**Insufficient room in root directory**
**Erase files from root and repeat CHKDSK**
(Non c'è spazio sufficiente nel directory radice
Cancellare alcuni file dalla radice e ripetere CHKDSK)

Quando CHKDSK ha tentato di ricreare file con i blocchi di dati "perduti", ha trovato il directory pieno. Dovete semplicemente eliminare alcuni di questi file ed eseguire nuovamente CHKDSK per recuperare tutti i blocchi di dati.

**Invalid sub-directory**
(Subdirectory non valido)

Il subdirectory indicato non è valido. Eseguite CHKDSK con l'opzione /V per ottenere maggiori informazioni al riguardo.

**Probable non-DOS disk. Continue (Y/N)?**
(Disco probabilmente non-DOS. Continuo (Y/N) ?)

Il disco non è riconoscibile come disco DOS. Continuate con molta cautela.

**Processing cannot continue**
(L'elaborazione non può continuare)

Messaggio parziale: è seguito dalla motivazione dell'impossibilità a continuare.

**Tree past this point not processed**
(La struttura ad albero non può essere seguita oltre a questo punto)

Ci sono problemi con la traccia numero 0 del disco che impediscono di determinare il resto del cammino.

**XXX bytes disk space freed**
(Liberati *XXX* byte di spazio su disco)

CHKDSK ha liberato *XXX* byte di blocchi "perduti".

**XXX lost clusters found in YYY chains**
(Trovati *XXX* blocchi perduti nella catena *YYY*)

CHKDSK ha trovato *XXX* blocchi di dati "perduti"; verrà richiesto un comando per renderli liberi.

## COMP

**Compare error at offset XXX**
(Errore di confronto nell'offset *XXX*)

I due file che vengono confrontati differiscono nell'offset *XXX* (esadecimale). Vengono visualizzati i valori esadecimali dei due byte.

**Compare more files (Y/N)?**
(Devo confrontare altri file (Y/N)?)

Vi consente di confrontare altri due file: se rispondete affermativamente, vi richiede i nomi dei nuovi file.

**Enter primary file name**
(Inserire il nome del primo file)

Inserire la specificazione del primo file.

**Enter 2nd file name or drive id**
(Inserire il nome del secondo file o l'identificatore del drive)

Inserire il nome del secondo file da confrontare. Se usate solo l'identificatore di drive (o la definizione di default premendo solo il tasto ENTER) verrà utilizzato ancora il nome del primo file.

**EOF mark not found**
(EOF non trovato)

Non è stato trovato nessun carattere EOF (end-of-file, codice esadecimale 1A) al termine di uno dei file. Ciò può accadere solo nel caso in cui vengono confrontati due file di tipo testo.

**File AND File**
(File AND File)

Visualizza le specificazioni dei due file che sono confrontati.

**Files are different sizes**
(I file hanno dimensioni diverse)

I due file da confrontare non sono della stessa dimensione: non viene eseguito alcun confronto.

**Files compare OK**
(Confronto file OK)

I due file che sono stati confrontati sono identici.

**10 Mismatches - ending compare**
(10 errori - fine del confronto)

COMP ferma il confronto dopo aver trovato 10 corrispondenze errate.

## COPY

**Cannot do binary reads from a device**
(Non potete effettuare la lettura binaria da un dispositivo)

Non potete usare l'opzione /B per copiare quando indicate un dispositivo, ad esempio la tastiera, come sorgente dei dati.

**Invalid path or file name**
(Nome del file o cammino non validi)

Avete indicato un directory o un nome di file che non esiste.

## CTTY

**Invalid device**
(Dispositivo non valido)

Il nome del dispositivo indicato non è valido.

## DATE

**Invalid date**
(Data non valida)

Sono stati inseriti una data o un delimitatore non validi; i delimitatori utilizzabili sono il trattino (-) o la barra (/).

## DEBUG

**Access denied**
(Accesso rifiutato)

Avete cercato di scrivere su di un file a sola lettura.

**BF**
(Bad Flag = flag errato)

Indicazione non valida del flag di registro con il comando R (Register).

**BP**
(Break Points = punti di interruzione)

Ci sono troppi punti di interruzione indicati nel comando GO: il limite massimo è dieci.

**BR**
(Bad Register = registro errato)

Il registro indicato nel comando non è valido.

**DF**
(Double Flag = doppio flag)

Una doppia indicazione di flag con un unico comando R.

**Error in EXE/HEX file**
(Errore nel file EXE/HEX)

Il file contiene record o caratteri non validi.

**EXE and HEX files cannot be written**
(I file EXE e HEX non possono essere scritti)

DEBUG non è in grado di scrivere nei file .EXE e .HEX.

**Insufficient space on disk**
(Lo spazio sul disco non è sufficiente)

Non c'è spazio libero sufficiente sul disco per contenere i dati scritti per mezzo del comando W (Write). Utilizzatene un altro con abbastanza spazio oppure uscite da DEBUG e cancellate alcuni file dal disco corrente.

## DISKCOMP

**Compare error(s) on Track XX, side YY**
(Errore di confronto sulla traccia *XX*, faccia *YY*)

Una o più locazioni, nella traccia e faccia indicate, non coincidono nei due dischetti da confrontare.

**Compare more diskettes (Y/N)?**
(Devo confrontare altri dischetti (Y/N) ?)

Rispondete affermativamente se volete confrontare altri due dischetti.

**Comparing X sectors per track, Y side(s)**
(Sto confrontando *X* settori per traccia, faccia/e *Y*)

Vi informa se sono confrontati otto settori o nove per traccia e una o due facce del dischetto.

**Diskettes compare OK**
(Corretto confronto tra i dischetti)

I dischetti appena confrontati sono identici.

**Incompatible diskette or drive types**
(Tipi di drive o di dischetto incompatibili)

Il primo dischetto è stato letto con successo, ma il secondo ha un formato differente: o il primo disco è a doppia faccia ed il secondo a singola, oppure il primo contiene nove settori per traccia mentre il secondo otto.

**Incompatible drive types**
(Tipi di drive incompatibile)

Il primo disco ed il primo drive sono a doppia faccia, mentre il secondo drive ha capacità di leggere e scrivere un solo lato.

**Insert first diskette in drive X**
**Insert second diskette in drive Y**
(Inserire il primo dischetto nel drive *X*
Inserire il secondo dischetto nel drive *Y*)

DISKCOMP vi richiede di porre i dischetti da confrontare nei drive indicati. Premendo un qualsiasi tasto, purché diverso dalla sequenza CTRL BREAK che invece interrompe l'attività di DISKCOMP, si fa riprendere l'azione.

**Unrecoverable read error on drive X. Track YY, side Z**
(Errore di lettura irreparabile nel drive *X*. Traccia *YY*, faccia *Z*)

I dati non possono essere letti dal drive indicato nella traccia specificata.

## DISKCOPY

**Copy another (Y/N)?**
(Un altro disco da copiare (Y/N) ?)

Rispondete affermativamente per fare un'altra copia da disco a disco.

**Copy complete**
(Copia completata)

La copia da disco a disco è stata completata con successo.

**Copying X sectors per track, Y side(s)**
(Sto copiando *X* settori per traccia, faccia/e *Y*)

Vi informa che la copia del disco sta utilizzando otto o nove settori per traccia e una o due facce del disco.

**Formatting while copying**
(Formattazione insieme alla copia)

Il disco destinazione aveva un formato non riconoscibile dal DOS, perciò DISKCOPY formatta il dischetto mentre effettua la copia da disco a disco.

**Insert source diskette in drive X**
**Insert target diskette in drive Y**
(Inserire il disco sorgente nel drive *X*
Inserire il disco destinazione nel drive *Y*)

Vi richiede di inserire i dischi da e su cui copiare negli opportuni drive. Per continuare, premete un tasto qualsiasi, tranne CTRL BREAK che fa terminare DISKCOPY.

**Target diskette may be unusable**
(Il disco destinazione potrebbe essere inutilizzabile)

È stato trovato un errore non rimediabile di lettura, di scrittura o di verifica. La validità del disco destinazione è sospetta. Utilizzate DISKCOMP per confrontare sorgente e destinazione.

**Target diskette write-protected. Correct, then strike any key**
(Il disco destinazione è protetto dalla sovrascrittura. Correggete e poi premete un qualsiasi tasto)

Il disco destinazione non può essere scritto se la tacca è coperta; potete uscire dal DISKCOPY premendo CTRL BREAK.

**Unrecoverable format error on target. Target diskette unusable**
(Errore di formato irreparabile nel disco destinazione; questo disco è inutilizzabile)

Il disco destinazione non è stato formattato in modo corretto: i dati contenuti nel disco non sono validi.

**Unrecoverable read error on source. Track XX, side Y**
(Errore irreparabile di lettura nel disco sorgente. Traccia *XX*, faccia *Y*)

Non è possibile leggere alcune parti del disco sorgente; la copia può non essere attendibile.

**Unrecoverable verify error on target. Track XX, side Y**
(Errore irreparabile di verifica nel disco destinazione. Traccia *XX*, faccia *Y*)

È stata tentata la verifica dei dati scritti sul dischetto, ma questa non può essere completata. I dati sul disco destinazione sono incompleti.

**Unrecoverable write error on target. Track XX, side Y**
(Errore irreparabile di scrittura sul disco destinazione. Traccia *XX*, faccia *Y*)

Parte dei dati non può essere scritta sul disco destinazione.

## DOS

**Bad command of file name**
(Comando o nome di file errato)

Il comando che avete inviato è un comando DOS errato, oppure un comando esterno o un file batch che non è stato trovato sul disco di default.

**Bad or missing Command Interpreter**
(Interprete dei comandi errato o mancante)

Il file DOS chiamato COMMAND.COM non è stato trovato sul disco, oppure si è verificato un errore di lettura nel caricamento dello stesso.

**Bad or missing <filename>**
(<*nomefile*> errato o mancante)

Il driver del dispositivo *nomefile* è stato indicato nell'istruzione DEVICE=*nomefile* in un file CONFIG.SYS, ma il file *nomefile* non è stato trovato. Il driver non è stato installato dal DOS.

**Batch file missing**
(File batch mancante)

Il DOS non è in grado di localizzare il file batch da elaborare.

**Cannot load COMMAND, system halted**
(Non posso caricare COMMAND; il sistema è fermo)

Il file COMMAND.COM non può essere localizzato. Riavviate il DOS con una reinizializzazione del sistema o con una riaccensione della macchina.

**Cannot start COMMAND, exiting**
(Non posso far partire COMMAND; esco dal DOS)

Il DOS non riesce a caricare ed eseguire COMMAND.COM, oppure non c'è sufficiente spazio in memoria per farlo.

**Disk boot failure**
(Errore nell'inizializzazione di un disco)

Si è verificato un errore nel caricare il DOS in memoria. Può darsi che dobbiate utilizzare un'altra copia del disco DOS per avviare il sistema.

**Divide overflow**
(Overflow causato da una divisione)

Un programma ha tentato di dividere un numero per zero, oppure contiene un errore logico. L'esecuzione del programma termina.

**Error in EXE file**
(Errore in un file EXE)

È stato trovato un errore nelle informazioni di rilocamento poste nel file EXE dal programma LINK. Non è in grado di caricare ed eseguire il file.

**Error writing to device**
(Errore nella scrittura destinata ad un dispositivo)

Il DOS non è in grado di completare la scrittura richiesta nel dispositivo indicato. Può darsi che abbiate inviato al dispositivo più dati di quanti ne attendesse.

**EXEC failure**
(Errore in EXEC)

È stato trovato un errore nel leggere un comando da disco, oppure il comando FILES= nel file CONFIG.SYS non indicava un valore sufficientemente grande.

**File allocation table bad, drive X**
**Abort, Retry, Ignore?**
(Errata tabella di allocazione dei file, drive X
Esco, Riprovo, Ignoro ?)

Errore su disco. Battete A (Abort) per terminare l'esecuzione del comando, R (Retry) per riprovare l'accesso al disco, oppure I (Ignore) per continuare come se l'errore non si fosse verificato. Se l'errore si dovesse ripetere, la causa potrebbe essere che il disco non è più utilizzabile e deve essere riformattato.

**File cannot be copied onto itseft**
(Il file non può essere copiato su se stesso)

Dovete indicare una specificazione per la destinazione differente dalla sorgente nella copia dei file.

**File creation error**
(Errore nella creazione di un file)

Qualche errore ha impedito la creazione o la modifica di un file o di un directory; potrebbe trattarsi di un file a sola lettura oppure di un directory pieno.

**File not found**
(File non trovato)

Il file indicato nel comando non è stato trovato nel directory e nel drive nominati.

**Incorrect DOS version**
(Versione del DOS non corretta)

Il comando richiede una versione del DOS differente da quella in esecuzione.

**Insert COMMAND.COM disk in drive X: and strike any key when ready**
(Inserire il disco COMMAND.COM nel drive *X*; premere un tasto qualsiasi quando tutto è pronto)

Il DOS ha cercato di usare COMMAND.COM, ma non ha potuto trovarlo nel drive di partenza. Inserite il disco del DOS e premete un tasto qualsiasi.

**Insert disk with batch file and strike any key when ready**
(Inserire il disco con il file batch e poi premere un tasto qualsiasi)

Il DOS non riesce a trovare il file batch da elaborare. Dovete inserire il disco contenente il file batch nel drive opportuno e premere un tasto qualsiasi per continuare. (Premete invece CTRL BREAK per far terminare il programma).

**Insufficient disk space**
(Spazio su disco non sufficiente)

Il DOS non può completare la scrittura su un disco a causa della mancanza di spazio. Potete eseguire CHKDSK con l'opzione /F per liberarlo dai blocchi inutilizzati o "perduti".

**Insufficient memory**
(Memoria insufficiente)

La quantità di memoria disponibile è troppo piccola.

**Intermediate file error during pipe**
(Errore in un file intermedio di un pipe)

Il DOS ha creato due file temporanei nel directory radice del drive di default nell'elaborare un comando di un pipe. Questo errore si verifica se il directory o il disco sono pieni, oppure se il DOS non è in grado di trovare i file da usare nel pipe. Dovete liberare una parte di disco per questi file temporanei.

**Invalid COMMAND.COM in drive X**
(COMMAND.COM non valido nel drive *X*)

Una versione non corretta di COMMAND.COM è stata trovata nel file letto dal disco. Inserite un dischetto DOS con la versione corretta e premete un tasto qualsiasi per continuare.

**Invalid directory**
(Directory non valido)

Uno dei directory indicati nel cammino non esiste.

**Invalid drive in search path**
(Drive non valido nella ricerca del cammino)

Nel cammino è stato indicato un drive non valido o inesistente.

**Invalid drive specification**
(Specificazione non valida di un drive)

È stata trovata un'indicazione non esatta di un drive.

**Invalid number of parameters**
(Numero di parametri non valido)

Avete indicato troppi o troppo pochi parametri per il comando.

**Invalid parameter**
(Parametro non valido)

È stato trovato un parametro non valido per quel comando.

**Memory allocation error. Cannot load COMMAND, system halted**
(Errore di allocazione della memoria, non è possibile caricare COMMAND; il sistema è fermo)

Una parte della memoria interna del DOS ha subito una sovrascrittura. Dovete riavviare il DOS per continuare.

**No free file handles. Cannot start COMMAND, exiting**
(Non c'è più possibilità di gestire i file. Non posso caricare COMMAND, esco dal DOS)

Ci sono già troppi file aperti per permettere di caricare una seconda copia di COMMAND.COM. Potete usare il comando FILES= nel file CONFIG.SYS per aumentare il numero di file che possono essere aperti contemporaneamente.

**Out of environment space**
(Manca spazio per l'environment)

Il DOS non è in grado di accettare il comando SET appena inviato poiché non è rimasto spazio sufficiente in memoria.

**Program too big to fit in memory**
(Programma troppo grande per essere contenuto nella memoria)

La quantità di memoria rimasta libera non è sufficiente per contenere il programma che state cercando di caricare o di eseguire.

**Syntax error**
(Errore di sintassi)

Il comando inviato non è nella forma corretta. Controllate la sintassi di questo comando.

**Terminate batch job (Y/N)?**
(Devo terminare il lavoro in batch (Y/N)?)

Richiesta ricevuta dopo aver premuto i tasti CTRL BREAK durante l'esecuzione di un file batch. Rispondete affermativamente per terminare il lavoro in batch, negativamente per continuare.

## EDLIN

**Cannot edit .BAK file – rename file**
(Non è possibile modificare un file con estensione .BAK — cambiare il nome del file)

Non potete modificare un file di backup.

**Disk full – write not completed**
(Il disco è pieno – la scrittura non è stata completata)

Non c'è spazio libero sufficiente sul dischetto per salvare il file per mezzo del comando E (End). La parte di file in memoria viene perduta.

**Entry error**
(Dato errato)

Sintassi non corretta nel comando precedente.

**Line too long**
(Linea troppo lunga)

Il limite per una linea di testo è di 253 caratteri. Il comando R (Replace) appena inviato ha superato questo limite.

**Must specify destination line number**
(Bisogna indicare il numero di linea della destinazione)

Un comando M (Move) o C (Copy) deve contenere anche un numero di linea valido per indicare la destinazione del comando stesso.

**No room in directory for file**
(Non c'è posto nel directory per questo file)

Il directory del disco indicato, o di default se non indicato, è pieno. Tutte le modifiche sono andate perse.

**Not enough room to merge the entire file**
(Non c'è spazio per fondere l'intero file)

Non c'è spazio in memoria sufficiente per trasferire interamente il contenuto del file indicato.

**Not found**
(Non trovato)

Il comando R (Replace) o S (Search) inviato non ha trovato nemmeno un'occorrenza della stringa indicata.

## EXE2BIN

**Amount read less than size in header**
(La parte letta è minore delle dimensioni specificate dall'header)

La parte del programma contenuto nel file è più piccola di quanto indicato nell'header del file. Bisogna ricompilare o riassemblare il file e poi linkarlo di nuovo.

**Fixups needed — base segment (hex)**
(Servono i fixup — segmento di base (esadecimale))

Il file sorgente (EXE) conteneva indicazioni sul fatto che era richiesto un segmento load per il file. Indicate l'indirizzo assoluto del segmento in cui il modulo, una volta terminato, dev'essere caricato.

**WARNING — Read error on EXE file**
(Attenzione — Errore di lettura in un file EXE)

Si è verificato un errore durante la lettura del file di input. Il file risultante può essere inutilizzabile.

## FDISK

**Do you wish to use the entire fixed disk for DOS (Y/N)?**
(Volete utilizzare l'intero disco rigido per il DOS (Y/N) ?)

Se rispondete affermativamente, l'intero disco rigido sarà riservato per il DOS, altrimenti vi verranno richiesti i limiti per la partizione da riservare al DOS.

**Enter partition size: [dddd]**
(Inserire le dimensioni della partizione: [*dddd*])

Prompt per la partizione del DOS; *dddd* indica la dimensione di default che può essere scelta premendo semplicemente ENTER.

**Enter starting cylinder number: [dddd]**
(Inserire il numero del cilindro di partenza: [*dddd*])

Cilindro in cui inizia la partizione del DOS; il cilindro di default è *dddd* e può essere richiesto premendo ENTER.

**Enter the number of the partition you want to make active:**
(Inserire il numero di partizione che volete rendere attiva)

Richiesta del numero di partizione da rendere attiva.

**Error reading fixed disk**
(Errore nella lettura del disco rigido)

FDISK non è in grado di leggere il record iniziale del disco rigido. Riprovate più volte e, nel caso, rivolgetevi al rivenditore.

**Error writing fixed disk**
(Errore nella scrittura del disco rigido)

FDISK non è in grado di scrivere il record iniziale del disco rigido; riprovate più volte e, nel caso, rivolgetevi al rivenditore.

**Fixed disk already has a DOS partition**
(Il disco rigido ha già una partizione riservata al DOS)

Avete richiesto la creazione di una partizione DOS per un disco che già ne conteneva una.

**Insert DOS diskette in drive A:**
**press any key when ready...**
(Inserite il dischetto con il DOS nel drive A:,
premete un tasto qualsiasi quando siete pronti)

La partizione riservata al DOS è predisposta sul disco rigido: inserendo il disco contenente il DOS nel drive A: e premendo un qualsiasi tasto inizializzate il sistema. Dovete ora formattare (FORMAT) il disco rigido.

**Maximum available space is XXXX cylinders at cylinder YYYY**
(Il massimo spazio disponibile è di *XXXX* cilindri a partire dal cilindro *YYYY*)

Indica il più grande spazio contiguo che si trova nel disco rigido.

**No DOS partition to delete**
(Non ci sono partizioni del DOS da cancellare)

Avete cercato di eliminare dal disco rigido una partizione DOS, ma non ne erano presenti.

**No fixed disks present**
(Non sono presenti dischi rigidi)

Il vostro sistema non possiede un disco rigido, oppure questo è installato in un'unità di sistema non alimentata, oppure non è installato correttamente. FDISK non può essere eseguito finché la situazione non viene modificata.

**No partitions to make active**
(Non ci sono partizioni da rendere attive)

Non potete modificare le partizioni del disco rigido perché non sono state trovate partizioni attive. Utilizzate l'opzione per creare una partizione riservata al DOS.

**No space for a XXXX cylinder partition**
(Non c'è spazio per una partizione di *XXXX* cilindri)

Avete richiesto una partizione più grande del più grande segmento libero nel disco. Utilizzate un numero più piccolo.

**No space for a XXXX cylinder partition at cylinder YYYY**
(Non c'è spazio per una partizione di *XXXX* cilindri a partire dal cilindro *YYYY*)

Avete richiesto una partizione per il DOS in un punto del corrente disco rigido in cui non c'è spazio sufficiente.

**No space to create a DOS partition**
(Non c'è spazio per creare una partizione per il DOS)

Avete richiesto l'opzione "di creazione" per il disco rigido corrente in cui non c'è spazio sufficiente per la partizione riservata al DOS.

**The current active partition is X**
(La partizione attiva corrente è *X*)

L'opzione per cambiare la partizione attiva visualizza quale sia la partizione attiva nel disco rigido corrente.

**Total disk space is XXXX cylinders**
(Lo spazio totale è di *XXXX* cilindri)

Lo spazio totale nel disco rigido corrente è di *XXXX* cilindri.

**Warning! All data in the DOS partition will be DESTROYED. Do you wish to continue? [N]**
(Attenzione! Tutti i dati della partizione del DOS andranno distrutti. Volete continuare? [N])

L'opzione per cancellare la partizione del DOS vi avvisa che se continuate perderete tutti i dati che si trovano in quella partizione del disco rigido. Se voi premete solo ENTER la partizione del DOS NON sarà distrutta; se volete proprio eliminarla dovete rispondere affermativamente alla domanda (Y e poi ENTER).

**X is not a choice. Enter a choice**
(*X* non è una scelta. Inserite una nuova scelta)

Avete risposto *X*, che non è una risposta ammissibile per la domanda proposta.

**X ist not a choice. Enter Y or N.**
(*X* non è una scelta. Rispondete Y o N)

Avete risposto *X*, che non è una risposta valida per la domanda in questione; potete rispondere solo affermativamente o negativamente.

## FORMAT

**Attempted write-protect violation**
(Avete tentato di violare una protezione contro la sovrascrittura)

Non potete formattare un dischetto coperto sulla tacca di protezione dalla sovrascrittura. Sistemate il dischetto e premete un qualsiasi tasto per far ripartire la formattazione.

**Disk not compatible**
(Il disco non è compatibile)

Il drive indicato non è supportato dal comando FORMAT.

**Disk unsuitable for system disk**
(Il disco non è utilizzabile come disco di sistema)

È stata trovata una traccia difettosa dove dovevano risiedere i file DOS

per il disco di sistema. Utilizzate questo disco solo per memorizzare dati o programmi.

**Format failure**
(Errore di formattazione)

L'errore avvenuto durante la formattazione rende il disco inutilizzabile.

**Insert DOS disk in X: and strike any key when ready**
(Inserite il disco DOS nel drive *X*: e premete un qualsiasi tasto quando siete pronti)

FORMAT ha bisogno dei file del DOS ma non li trova nel drive *X*:. Inserite il corretto disco nel drive e premete un tasto qualsiasi per continuare.

**Invalid characters in volume label**
(Caratteri non validi nella volume label)

La sintassi della volume label non è corretta: è stato trovato un carattere non valido.

**Parameter not compatible with fixed disk**
(Il parametro non è compatibile con il disco rigido)

Non potete usare le opzioni /1 e /8 con il comando FORMAT quando lo utilizzate per il disco rigido.

**Parameters not compatible**
(I parametri non sono compatibili)

Avete indicato due opzioni con parametri non compatibili nella linea del comando FORMAT.

**Press any key to begin formatting X**
(Premete un tasto qualsiasi per iniziare la formattazione del disco *X*)

Premete un qualsiasi tasto per dare avvio alla formattazione del disco rigido *X*. Usate CTRL BREAK se non volete formattarlo. NOTA: la formattazione cancella tutti i dati che risiedevano precedentemente sul disco!!

**Track 0 bad — disk unusable**
(La traccia 0 è difettosa — il disco è inutilizzabile)

La traccia 0 è il punto in cui devono risiedere il record di inizializzazione

di disco, la tabella di allocazione dei file ed il directory: se questa è difettosa, il disco risulta inutilizzabile.

**Unable to write BOOT**
(Non sono in grado di scrivere BOOT)

La prima traccia del dischetto o della partizione del DOS di un disco rigido è difettosa; il record di avviamento non può essere scritto.

**Volume label (11 characters, ENTER for none)?**
Volume label (11 caratteri, ENTER per nessuna)?

Richiesta di inserimento della volume label per il disco appena formattato.

## LINK

**About to generate .EXE file**
**Change disks <hit ENTER>**
(Per generare un file .EXE cambiate dischi e premete ENTER)

Inserite il vostro disco che deve contenere il programma finale, dopo aver indicato il parametro /Pause.

**Ambiguous switch: z**
(Indicatore ambiguo: *z*)

Il carattere indicato da *z* non identifica univocamente un parametro di link. Utilizzate un numero maggiore di caratteri per il nome del parametro.

**An internal failure has occurred**
(Si è verificato un errore interno)

Si è verificato un errore nel linker. Riferite le condizioni in cui il messaggio è apparso al vostro rivenditore.

**Attempt to access data outside of segment bounds**
(Tentativo di accedere a dati che si trovano all'esterno dei limiti del segmento)

Un file oggetto è probabilmente non valido.

**Bad numeric parameter**
(Parametro numerico errato)

Il valore indicato nel parametro /STACK non è una costante numerica valida.

**Cannot find file filespec**
**Change diskette<hit ENTER>**
(Non posso trovare il file *specfile*
Cambiate il dischetto <premete ENTER>)

Il linker non è in grado di trovare il modulo oggetto *specfile* nel drive indicato. Inserite il dischetto contenente il modulo richiesto e premete ENTER.

**Cannot find library libname**
**Enter new drive letter**
(Non posso trovare la libreria *nomelib*
Inserite una diversa lettera per il drive)

Il file contenente la libreria indicata non si trova nel drive. Inserite il nome del drive in cui è possibile trovare il file.

**Cannot nest response file**
(Non posso annidare il file di risposta)

Avete utilizzato una specificazione all'interno di un file di risposta automatica: questi file non possono essere annidati.

**Cannot open list file**
(Non posso aprire il file list)

Il directory o il disco sono pieni.

**Cannot open overlay**
(Non posso aprire un overlay)

Il directory o il disco sono pieni.

**Cannot open response file**
(Non posso aprire un file di risposta)

Il file di risposta automatica non è stato trovato.

**Cannot open temporary file**
(Non posso aprire un file temporaneo)

Il directory o il disco sono pieni.

**Dup record too complex**
(Record DUP troppo complesso)

Il problema risiede in un modulo oggetto creato da un programma sorgente in Assembler. Un singolo DUP richiede 1024 byte prima dell'espansione. Sottoponete al debug il programma sorgente in Assembler e poi rieseguite LINK.

**Fixup offset exceeds field width**
(Gli offset del fixup eccedono la larghezza del campo)

Un'istruzione per il processore in linguaggio macchina si riferisce ad un indirizzo con un attributo NEAR (vicino) invece che FAR (lontano). Modificate (con Edit) il programa sorgente in Assembler e rielaboratelo.

**Invalid format file**
(File con formato non valido)

Un file di libreria presenta un errore.

**Invalid numeric parameter**
(Parametro numerico non valido)

Un valore numerico non è espresso in cifre.

**Invalid object module**
(Modulo oggetto non valido)

Un modulo oggetto è formato in modo non corretto oppure è incompleto (come accade quando il processore del linguaggio viene fermato nel mezzo dell'elaborazione).

**Invalid switch: z**
(Indicatore non valido: *z*)

Il carattere indicato da *z* non rappresenta un parametro valido per LINK.

**No object modules specified**
(Non sono stati indicati moduli oggetto)

Non avete nominato alcun modulo oggetto nella linea di comando o in risposta al prompt. Il LINK ha bisogno del nome del file.

**Out of space on list file**
(Spazio mancante per il file List)

L'errore si verifica normalmente quando non c'è spazio sufficiente per il file List.

**Out of space on run file**
(Spazio mancante per il file di esecuzione)

Questo errore accade di solito quando non c'è spazio sufficiente sul disco per il file (.EXE).

**Out of space on VM.TMP**
(Spazio mancante per VM.TMP)

Non rimane spazio sul disco per espandere il file VM.TMP.

**Program size exceeds capacity of LINK**
(Le dimensioni del programma eccedono le capacità di LINK)

Il modulo caricato è troppo grande per l'elaborazione.

**Requested stack size exceeds 64K**
(Le dimensioni richieste per lo stack superano i 64K)

Indicate una dimensione minore di 64 Kbyte quando compare la richiesta per la dimensione dello stack.

**Segment size exceeds 64k**
(Le dimensioni del segmento superano i 64K)

Un tentativo di combinare segmenti con lo stesso nome si è risolto con un segmento più grande di 64 Kbyte. Il limite per gli indirizzi è di 64 Kbyte.

**Stack size exceeds 65535 bytes**
(La dimensione dello stack supera 65535 byte)

Le dimensioni indicate per lo stack devono essere inferiori o al massimo uguali a 65535 byte.

**Symbol defined more than once**
(Simbolo definito più di una volta)

Il linker ha trovato due o più moduli che definiscono lo stesso simbolo.

**Symbol table capacity exceeded**
(La capacità della tabella dei simboli è stata superata)

Sono stati utilizzati troppi nomi e troppo lunghi; questi superano all'incirca i 50 Kbyte. Usate nomi più corti o meno nomi, oppure entrambe le cose.

**Too many external symbols in one module**
(Troppi simboli esterni in un modulo)

Il limite è di 256 simboli esterni per ogni modulo.

**Too many groups**
(Troppi gruppi)

Il limite è di dieci gruppi, compreso DGROUP.

**Too many libraries specified**
(Troppe indicazioni di librerie)

Il limite è di otto.

**Too many overlays**
(Troppi overlay)

Il limite è di 64.

**Too many public symbols in one module**
(Troppi simboli comuni in un modulo)

Il limite è di 1024 simboli comuni.

**Too many segments or classes**
(Troppi segmenti o classi)

Il limite è di 247 (segmenti e classi insieme).

**Unexpected end-of-file on library**
(End-of-file inatteso in una libreria)

Probabilmente causato da un errore nel file.

**Unexpected end-of-file on VM.TMP**
(End-of-file inatteso in VM.TMP)

Il dischetto che contiene VM.TMP è stato tolto dal drive.

**Unresolved externals: list**
(Simboli esterni non risolti: lista)

I simboli esterni elencati non erano stati definiti nei file modulo o libreria che voi avete indicato. Se si verifica questo errore, non cercate di eseguire il file eseguibile creato dal linker.

**VM.TMP is an illegal filename and has been ignored**
(VM.TMP è un nome illegale ed è stato ignorato)

Non potete usare VM.TMP come nome per un file oggetto. Questo messaggio è solo un avviso.

**Warning: no stack segment**
(Attenzione: non c'è nessun segmento stack)

Nessuno dei moduli oggetto indicato contiene uno spazio per lo stack in cui inserire le istruzioni.

## MKDIR

**Unable to create directory**
(Non sono in grado di creare un directory)

Il directory da voi indicato esiste già, oppure uno dei directory del cammino non è stato trovato, o anche il directory radice è pieno.

## MODE

**COMn: bbbb,p,d,s,t initialized**
(COM*n: bbbb,p,d,s,t* inizializzato)

L'interfaccia per comunicazioni asincrone (ACA) è stata inizializzata. I valori indicati rappresentano:

*n* interfaccia (1 o 2)
*bbbb* baud rate (110, 150, 300, 600, 1200, 2400, 4800 o 9600)

*p* parità (e - pari, o - dispari, n - nessuna)
*d* bit di dati (7 o 8, default=7)
*s* bit di stop (1 o 2)
*t* tipo di dispositivo seriale: p (stampante seriale, in cui agli errori di timeout si ripete la richiesta) oppure — (altri dispositivi seriali, in cui ciò non accade)

**Do you see the leftmost 9? (Y/N)**
(Vedete il 9 più a sinistra (Y/N) ?)

Avete indicato MODE ,R,T. Rispondendo N la videata si sposterà ancora a destra.

**Do you see the rightmost 9? (Y/N)**
(Vedete il 9 più a destra (Y/N)?)

Avete indicato MODE ,L,T. Rispondendo N la videata si sposterà ancora a sinistra.

**Illegal Device Name**
(Nome di dispositivo illegale)

Il dispositivo indicato dev'essere uno tra questi: LPT1:, LPT2:, LPT3:, COM1:, COM2:.

**Invalid baud rate specified**
(Il baud rate indicato non è valido)

Indicate i primi due, o più caratteri di uno dei seguenti valori permessi: 110, 150, 300, 600, 1200, 2400, 4800, 9600.

**Invalid parameters**
(Parametri non validi)

Non sono stati inseriti parametri nel comando MODE, o il primo parametro era diverso da L, C, 40, 80, BW40, BW80, CO40, CO80 o MONO, oppure l'interfaccia video indicata non è presente nel sistema.

**LPT# not redirected**
(LPT# non è stata reindirizzata)

I caratteri inviati alla stampante parallela non saranno reindirizzati ad un altro dispositivo.

**LPT#: redirected to COMn:**
(LPT#: reindirizzata a COM*n*:)

Tutti i caratteri che in output sono destinati alla stampante parallela #(# =1, 2 o 3) vengono reindirizzati al dispositivo seriale *n* (*n*=1 o 2).

**LPT#: set for 80**
(LPT#: predisposta per 80 caratteri)

La stampante parallela # (# =1, 2 o 3) è predisposta con una larghezza di linea di 80 caratteri.

**LPT#: set for 132**
(LPT#: predisposta per 132 caratteri)

La stampante parallela # (# =1, 2 o 3) è predisposta con una larghezza di linea di 132 caratteri.

**Printer error**
(Errore della stampante)

Non è in grado di predisporre il modo richiesto a causa di un errore di I/O, della mancanza di carta nella stampante, della mancata alimentazione della stampante oppure di una condizione di timeout.

**Printer lines per inch set**
(Predisposizione numero di linee di stampa per pollice)

Fissa la spaziatura verticale di stampa a sei o otto linee per pollice.

**Resident portion of MODE loaded**
(La parte residente di MODE è stata caricata)

Quando MODE viene utilizzata per altre funzioni diverse dalla definizione dello schermo, a volte è necessario caricare una parte di codice e renderlo residente.

## MORE

**-MORE-**
(-Ancora?-)

Il comando MORE ha inviato una pagina di dati al dispositivo standard

di output ed attende che premiate un tasto qualsiasi per inviare la successiva.

## PATH

**No path**
(Nessun cammino)

Non sono indicati cammini alternativi per il DOS affinché possa trovare comandi o file batch al di fuori del directory indicato.

## PRINT

**All files canceled by operator**
(Tutti i file sono stati cancellati dall'operatore)

Risposta alla cancellazione di tutti i file nella coda di stampa per mezzo del parametro /T PRINT.

**Errors on list device indicate that
it may be off-line. Please check.**
(Errori nel dispositivo list indicano che questo potrebbe essere spento o non collegato. Controllate)

Controllate il dispositivo per capire perché non risponde all'output della stampante.

**File canceled by operator**
(File cancellato dall'operatore)

Risposta inviata alla stampante dopo che un file da stampare è stato cancellato.

**File is currently being printed. File is in queue.**
(Il file è attualmente in stampa. Il file è in coda)

Risposta ad un comando PRINT senza parametri. Elenca i file che sono in attesa di essere stampati e quello attualmente in stampa.

**List output is not assigned to a device**
(L'output non corrisponde ad alcun dispositivo)

Il dispositivo indicato non corrisponde ad alcun dispositivo valido. Dovete usare di nuovo il comando PRINT con un nome valido per il dispositivo.

**Name of list device [PRN]:**
(Nome del dispositivo [PRN]:)

Questo messaggio compare la prima volta che usate PRINT. Rispondete con un nome valido di dispositivo al quale volete inviare l'output di PRINT. Premete ENTER se volete utilizzare la stampante parallela #1.

**Print queue is empty**
(La coda di stampa è vuota)

È la risposta al comando senza parametri, se nessun file è in stampa.

**Print queue is full**
(La coda di stampa è piena)

PRINT ha un limite di dieci file per la coda di stampa.

**Resident part of PRINT installed**
(È stata installata la parte di PRINT residente)

Il messaggio appare la prima volta che utilizzate il comando PRINT; PRINT utilizza all'incirca 3200 byte di memoria.

**XXX error on file YYY**
(Errore *XXX* nel file *YYY*)

Messaggio inviato alla stampante se si è verificato un errore su disco. La stampa del file termina.

## RECOVER

**Insert disk to be recovered into drive X:**
**and press any key when ready**
(Inserire il disco da recuperare nel drive *X:*
e premere un tasto qualsiasi)

Inserire il disco richiesto nel drive *X*: e premere un tasto per riprendere il processo di recupero.

**Warning — directory full**
(Attenzione — il catalogo è pieno)

Non c'è più spazio nel catalogo per recuperare altri file; eliminate alcuni dei file correnti prima di continuare.

## RENAME

**Duplicate filename or file not found**
(Nome di file duplicato oppure file non trovato)

Non potete chiamare un file con il nome di un altro file già esistente. Il messaggio appare anche nel caso in cui il file indicato non venga trovato.

**Missing file name**
(Manca il nome del file)

Non è stato specificato il nome del secondo dei due file.

## RESTORE

**Backup file sequence error**
(Errore nella sequenza dei file di backup)

Un file da ripristinare è stato copiato su più di un dischetto; il dischetto inserito non è il primo della sequenza. Eseguite di nuovo RESTORE con il disco corretto.

**Diskette is not a backup diskette**
(Il disco non è un disco di backup)

Il dischetto non è stato creato con il comando BACKUP. Ripetete con il disco giusto.

***** Files were backed up mm/dd/yyyy *****
(*** I file sono stati copiati il *mm/gg/aa* ***)

Data in cui i file da ripristinare erano stati copiati.

**Insert backup diskette XX in drive Y:**
**Strike any key when ready**
(Inserite il dischetto di backup *XX* nel drive *Y*:
Premete un tasto qualsiasi per continuare)

Inserite il successivo dischetto della sequenza. Premendo un qualsiasi tasto l'esecuzione riprende.

*** * * Restoring files from diskette XX * * ***
(* * * File ripristinati dal dischetto *XX* * * *)

Elenco dei file ripristinati dal dischetto indicato.

**The last file was not restored**
(L'ultimo file non è stato ripristinato)

Avete fermato RESTORE prima che avesse terminato con l'ultimo file, oppure non c'era più spazio per questo sul disco rigido.

**Warning! Diskette is out of sequence**
**Replace the diskette or continue**
**Strike any key when ready**
(Attenzione! Il dischetto non è nel corretto ordine di sequenza
Sostituite il dischetto oppure continuate
Premete un tasto qualsiasi quando siete pronti)

Il dischetto non è in sequenza. Potete continuare anche così se non vi interessa nessuno dei file che avete saltato con il cambiamento della sequenza.

**Warning! File XX is a read-only file.**
**Replace the file (Y/N)?**
(Attenzione! Il file *XX* è a sola lettura.
Devo sostituire il file (Y/N) ?)

Il file indicato è a sola lettura. Rispondete affermativamente se volete sostituirlo, altrimenti rispondete di no.

**Warning! File XX was changed after it was backed up.**
**Replace the file (Y/N)?**
(Attenzione! Il file *XX* è stato modificato dopo il backup. Devo sostituire il file (Y/N) ?)

Il file indicato nel disco rigido ha una data posteriore a quella della copia di backup. Rispondete di sì se volete sostituire la versione su disco rigido con quella del disco di backup, no altrimenti.

**Warning! No files were found to restore**
(Attenzione! Non sono stati trovati file da ripristinare)

Non ci sono file di backup sul dischetto.

## RMDIR

**Invalid path, not directory; or directory not empty**
(Cammino non valido, nessun directory; oppure directory non vuoto)

Il directory indicato non è stato rimosso dal disco perché parte del cammino indicato non era valida, oppure perché il directory da eliminare non era vuoto.

## SYS

**Incompatible system size**
(Dimensione incompatibile con il sistema)

Il DOS occupa più spazio di quanto ne sia stato riservato sul disco destinazione. Il trasferimento di sistema non avviene.

**Insert DOS disk in X: and strike any key when ready**
(Inserite il disco DOS nel drive *X*: e premete un qualsiasi tasto per continuare)

SYS ha bisogno dei file del DOS ma non li trova nel drive *X*:. Inserite il disco nel drive e premete un tasto per ripartire.

**No room for system on destination disk**
(Non c'è spazio per il sistema sul disco destinazione)

Il disco destinazione non è stato formattato con lo spazio riservato per il DOS con l'opzione /B di FORMAT. Il sistema non può essere trasferito.

## TIME

**Invalid time**
(Ora non valida)

Avete inserito un'ora o un delimitatore non valido; gli unici delimitatori validi sono i due punti (:) ed il punto (.) nel formato [*hh:mm:ss.xx*].

## TREE

**Invalid path**
(Cammino non valido)

C'è qualche errore nella struttura del catalogo. Usate CHKDSK per determinare il problema.

**No sub-directories exist**
(Non esistono subdirectory)

Nel disco indicato esiste solo il directory radice.

## Messaggi d'errore all'avviamento del sistema

**Error loading operating system**
(Errore nel caricamento del sistema operativo)

Si è verificato un errore del disco mentre stavate caricando il vostro sistema operativo dal disco rigido. Probabilmente dovete avviare il DOS con un dischetto e immagazzinare una nuova copia del DOS nel disco rigido.

**Invalid partition table**
(Tabella di partizioni non valida)

Sono state rilevate informazioni non valide riguardanti le partizioni del disco rigido. Avviate il DOS dal dischetto e correggete le informazioni.

**Missing operating system**
(Manca il sistema operativo)

Non è stata trovata una copia del sistema operativo nel disco rigido. Caricate il DOS da dischetto, copiate tutti i file sul disco rigido e riformattate il disco rigido utilizzando l'opzione /S in FORMAT.

**Non-System disk or disk error**
**Replace and strike any key when ready**
(Disco non contenente il sistema o errore del disco
Sostituitelo e premete un tasto qualsiasi quando siete pronti)

Uno dei due file nascosti IBMBIO.COM e IBMDOS.COM non è stato tro-

vato nel disco di avvio del sistema, oppure si è verificato un errore nella lettura del disco. Provate a caricare il DOS utilizzando il disco DOS nel drive A:.

**Sector size too large in file<filename>**
(Dimensione dei settori troppo grande nel file <*nomefile*>)

Il drive del dispositivo *nomefile* indica per il dispositivo un settore più grande di quello definito precedentemente dal DOS.

**Unrecognized command in CONFIG.SYS**
(Comando non riconosciuto in CONFIG.SYS)

Nel file di configurazione CONFIG.SYS è stato trovato un comando non valido. Dovete modificare il file, correggere il comando e riavviare il DOS.

# Indice analitico

La McGraw-Hill pubblica in tutto il mondo centinaia di libri di informatica per lo studio, la professione e il tempo libero. La produzione in lingua italiana comprende:

| | |
|---|---|
| 88 386 0001 5 | J. Heilborn e R. Talbott, *Guida al Commodore 64* |
| 88 7700 002 3 | C.A. Street, *La gestione delle informazioni con lo ZX Spectrum* |
| 88 7700 003 1 | T. Woods, *L'Assembler per lo ZX Spectrum* |
| 88 7700 004 X | R. Jeffries, G. Fisher e B. Sawyer, *Divertirsi giocando con il Commodore 64* |
| 88 7700 005 8 | G. Bishop, *Progetti hardware con lo ZX Spectrum* |
| 88 7700 006 6 | H. Mullish e D. Kruger, *Il BASIC Applesoft* |
| 88 7700 007 4 | N. Williams, *Progettazione di giochi d'avventura con lo ZX Spectrum* |
| 88 386 0008 2 | H. Peckham, *Il BASIC e il PC-IBM in pratica* |
| 88 7700 009 0 | H. Peckham, *Il BASIC e il Commodore 64 in pratica* |
| 88 7700 010 4 | S. Nicholls, *Tecniche avanzate in Assembler con lo ZX Spectrum* |
| 88 7700 011 2 | K. Skier, *L'Assembler per il Commodore 64 e il VIC-20* |
| 88 7700 012 0 | S. Kamins e M. Waite, *Programmazione umanizzata in Applesoft* |
| 88 7700 013 9 | A. Pennell, *Guida allo ZX Microdrive e all'Interface 1* |
| 88 7700 015 5 | P. Cohen, *Grafica e animazione con gli Apple II* |
| 88 7700 016 3 | C. Duff, *Guida al Macintosh* |
| 88 7700 017 1 | G. Kane, *Il manuale MC68000* |
| 88 386 0018 X | P. Hoffman e T. Nicoloff, *Il manuale MS-DOS* |
| 88 386 0019 8 | E.M. Baras, *Come usare il Symphony* |
| 88 7700 020 1 | S. Nicholls, *Grafica avanzata con lo ZX Spectrum* |
| 88 386 0021 X | L.J. Graham e T. Field, *Guida al PC-IBM* |
| 88 7700 022 8 | T. Field, *Come usare MacWrite e MacPaint* |
| 88 7700 024 4 | H. Peckham, *Il BASIC e gli Apple II in pratica* |
| 88 386 0025 2 | C. Morgan e M. Waite, *Il manuale 8086/8088* |
| 88 386 0026 0 | W. Ettlin, *Come usare il Multiplan* |
| 88 7700 027 9 | G. Mainis, *Il manuale ProDOS* |
| 88 7700 028 7 | J. Jones, *Il SuperBASIC del QL* |
| 88 7700 029 5 | C. Opie, *L'Assembler per il QL* |
| 88 386 0030 9 | W. Ettlin e G. Solberg, *Il BASIC Microsoft* |
| 88 386 0031 7 | D.L. Toppen, *Il Forth in pratica* |
| 88 7700 032 5 | R. Person, *Le meraviglie dell'animazione con gli Apple II* |
| 88 386 0033 3 | P.A. Sand, *Programmazione avanzata in Pascal* |
| 88 7700 034 1 | P. Hoffman, *Il manuale MSX* |
| 88 7700 035 X | R. Person, *Le meraviglie dell'animazione con il PC-IBM* |
| 88 7700 036 8 | W. Ettlin, *Il Multiplan per il Macintosh* |
| 88 386 0037 6 | D. Kruglinski, *Introduzione al Framework* |
| 88 386 0038 4 | L. Barnes, *Come usare il dBase II* |
| 88 386 0040 6 | L. Barnes, *Come usare il dBase III* |
| 88 386 0041 4 | W. Ettlin e G. Solberg, *Il GW-BASIC per Personal Computer Olivetti* |
| 88 386 0042 2 | D. Watt, *Il LOGO per il Commodore 64* |
| 88 386 0043 0 | T.J. Byers, *Guida al PC AT IBM* |
| 88 386 0044 9 | D. Kater e R. Kater, *Guida alle stampanti Epson* |
| 88 386 0045 7 | R.L. Tokheim, *Progetti di circuiti elettronici per microcalcolatori* |

| | |
|---|---|
| 88 386 0042 2 | D. Watt, *Il LOGO per il Commodore 64* |
| 88 386 0043 0 | T.J. Byers, *Guida al PC AT IBM* |
| 88 386 0044 9 | D. Kater e R. Kater, *Guida alle stampanti Epson* |
| 88 386 0045 7 | R.L. Tokheim, *Progetti di circuiti elettronici per microcalcolatori* |
| 88 386 0047 3 | C. Siechert e C. Wood, *Il manuale MS-DOS 3.20* |
| 88 386 0048 1 | H. Peckham, *Programmazione strutturata in BASIC* |
| 88 386 0049 X | W.H. Murray III e C.H. Pappas, *L'Assembler per l'80286/80386* |
| 88 386 0050 3 | D. Andersen, C. Cooper e B. Dempsey, *dBase III in pratica* |
| 88 386 0051 1 | D. Carroll, *Programmazione in Turbo Pascal* |
| 88 386 0052 X | P. Hoffman e T. Nicoloff, *Il manuale MS-DOS per Personal Computer Olivetti* |
| 88 386 0055 4 | J. Heilborn, *Guida al Commodore 128* |
| 88 386 0056 2 | E. Jones, *Come usare dBase III Plus* |
| 88 386 0057 0 | M. Guerriero e H. Zampariolo, *Programmare in FRED con Framework II* |
| 88 386 0060 0 | P. Robinson, *Programmazione in Turbo Prolog* |
| 88 386 0063 5 | A.E. Stanley, *Il BASIC 7.0 per il Commodore 128* |
| 88 386 0064 3 | W.A. Ettlin, *Come usare il WordStar 4.0* |
| 88 386 0601 3 | S. Harrington, *Computer Graphics - Corso di programmazione* |
| 88 386 0602 1 | O. Lecarme e J.L. Nebut, *Pascal - Guida per programmatori* |
| 88 386 0603 X | M. McGilton e R. Morgan, *Il sistema operativo UNIX* |
| 88 386 0604 8 | E. Rich, *Intelligenza Artificiale* |
| 88 386 0605 6 | M. Schagrin, J. Rapaport e R. Dipert, *Logica e computer* |
| 88 386 0606 4 | W. Newman e R. Sproull, *Principi di Computer Graphics* |
| 88 386 0607 2 | L. Hancock e M. Krieger, *Il linguaggio C* |
| 88 386 0615 3 | J.W.L. Ogilvie, *Il linguaggio Modula-2* |
| 88 386 0609 9 | S. Chapra e R. Canale, *Metodi numerici per l'ingegneria* |
| 88 386 0610 2 | P.R. Gray e R.G. Meyer, *Circuiti integrati analogici - Analisi e progetto* |
| 88 386 0608 0 | R. Thomas, L.R. Rogers e J.L. Yates, *UNIX System V Complementi di programmazione* |
| 88 386 0616 1 | R. Holloway, *Progettazione di strutture con i microcalcolatori* |

## La produzione software comprende:

| | |
|---|---|
| 88 7700 902 0 | C.A. Street, *PROFILE 2 - Foglio elettronico integrato per lo ZX Spectrum* |
| 88 7700 903 9 | S. Nicholls, *Routines in Assembler per la grafica avanzata con lo ZX Spectrum* |
| 88 7700 904 7 | A. Bleasby, *Assembler/Disassembler per il Commodore 64* |
| 88 7700 905 5 | ACS Software, *ZX Spectrum Monitor* |
| 88 7700 906 3 | G. Fitzgibbon, *Projector 1 - Uno strumento per costruire presentazioni grafiche con lo ZX Spectrum* |
| 88 386 0907 1 | C. Opie, *QL Machine Code Editor/Assembler* |
| 88 386 0908 X | B. Thompson e W. Thompson, *Sistema Esperto McGraw-Hill Versione 1.0 per personal MS-DOS* |

| | |
|---|---|
| 88 386 0909 8 | A. Tal, *Generatore di lezioni per il Commodore 64* |
| 88 386 0911 X | B. Thompson e W. Thompson, *Sistema Esperto McGraw-Hill per Apple II* |
| 88 386 0912 8 | D. Carroll, *Programmazione in Turbo Pascal* |
| 88 386 0917 6 | P. Lazzarini e G. Sarnataro, *CARTESIO-Microlinguaggio per lo studio delle trasformazioni geometriche* |
| 88 386 0918 7 | B. Thompson e W. Thompson, *Sistema Esperto McGraw-Hill Versione 2.0 per personal MS-DOS* |

Questo volume, sprovvisto del talloncino a fronte, è da considerarsi copia saggio e campione gratuito fuori commercio. Fuori campo applicazione IVA ed esente da bolla di accompagnamento (art. 22 L. 67/1987, art. 2 lett. i DPR 633/1972 e art. 4 n. 6 DPR 627/1978).

L. 54 000

Graham - Field
GUIDA AL PC-IBM
McGraw-Hill Libri Italia
88 386 0021 X

Il PC-IBM è sicuramente uno dei più completi personal computer presenti sul mercato: le caratteristiche del microprocessore, la possibilità di espansione, il potente sistema operativo, la ricca biblioteca di software disponibile e, non ultimo, il fascino legato all'immagine dell'azienda leader del settore, ne fanno un prodotto di vasta diffusione e soprattutto uno standard al quale gli altri sistemi devono fare continuo riferimento.
La *Guida al PC-IBM* è lo strumento necessario per chiunque voglia muoversi con sicurezza nella complessa struttura della macchina e acquisire le informazioni necessarie per sfruttarla compiutamente nel proprio settore di applicazione.
Si rivolge a tutti gli utenti indistintamente: sia agli esperti che vogliono trovare rapidamente la risposta ai loro quesiti, sia ai principianti che ricercano un valido testo per apprendere il funzionamento.
Tutti gli aspetti di questo potente personal computer sono trattati con estremo dettaglio:

- la struttura della macchina e i principi generali di funzionamento
- il sistema operativo PC-DOS, con tutte le caratteristiche della versione 2.1
- il linguaggio BASIC con la completa sintassi di tutti i comandi, le istruzioni e le funzioni
- la gestione dei file ad accesso diretto e sequenziali
- la grafica e il colore
- il suono
- l'interfacciamento con dispositivi esterni e le comunicazioni
- l'uso del disco rigido del PC/XT.

Da un originale  Osborne/McGraw-Hill


ISBN 88-386-0021-X
9 788838 600210

Lire 54 000

Graham
Field

Guida al PC IBM

021-X

